NightSplitter: a scheduling tool to optimize (sub)group activities

Tong Liu, Roberto Di Cosmo, Maurizio Gabbrielli, Jacopo Mauro

NightSplitter: a scheduling tool to optimize (sub)group activities

Tong Liu[1], Roberto Di Cosmo[2], Maurizio Gabbrielli[1] and Jacopo Mauro[3]

[1] DISI, University of Bologna, Bologna, Italy
[2] INRIA and University Paris Diderot, Paris, France
[3] Department of Informatics, University of Oslo, Oslo, Norway
{t.liu, maurizio.gabbrielli}@unibo.it, roberto@dicosmo.org, jacopom@ifi.uio.no

Abstract. Humans are social animals and usually organize activities in groups. However, they are often willing to split temporarily a bigger group in subgroups to enhance their preferences. In this work we present NightSplitter, an on-line tool that is able to plan movie and dinner activities for a group of users, possibly splitting them in subgroups to optimally satisfy their preferences. We first model and prove that this problem is NP-complete. We then use Constraint Programming (CP) or alternatively Simulated Annealing (SA) to solve it. Empirical results show the feasibility of the approach even for big cities where hundreds of users can select among hundreds of movies and thousand of restaurants.

1 Introduction

Nowadays, most of the city activities such as restaurants, cinemas, museums, theaters have complete and detailed information on web pages and offer a variety of online services and options for consulting programs, making reservations, buying tickets, etc. One of the main problems that the customer has to face in order to take advantage of this huge offer is to master the information overload which comes with it. For example, in Paris, our reference town for this work, there are more than 13500 restaurants and around 100 cinemas with 150 movies each night. Hence, the apparently simple task of organising a night out with a movie followed by a dinner can already turn into a serious planning exercise.

When there are several persons involved, e.g., a family or a group of friends, with different ideas, preferences, and needs, coordinating the activities of the group becomes significantly more complex. It is quite natural, in order to satisfy all the preferences of the members of a group, to take a pragmatic approach and split the group of persons into several sub-groups performing different activities, in order to enhance the individual satisfactions: some groups will watch the latest Hollywood blockbusters, while some others will prefer an indie movie, provided, of course, this can take place approximately at the same time, and in the same movie theater, or in movie theaters not too far apart.

And that's not all: one needs to take into account both time constraints (e.g., we need to be home before midnight) and spatial constraints (e.g., we do not

have the car and we do not want to walk for one hour). The planning of a night out can therefore easily become a daunting task.

Recommender systems and planners provide tools that can help users to manage these difficulties by filtering information, suggesting solutions, predicting some needs and planning the activities. However, most of the existing tools focus on a single user, so they cannot be used when several users interact and participate in a group activity [17,7]. Tools considering group experiences exist [3,5,20] but they mainly focus on methods for aggregating preferences for a fixed group of users in order to optimize (some notions of) group satisfaction.

Only a few research papers [4,18] consider the problem of sub-group formation and group splitting, but they do not take into account time and space constraints or they impose the same subgroups for all the activities, thus forbidding the most interesting cases, like a group that splits into subgroups to see different movies, but then joins at the same restaurant.

In this work we present `NightSplitter`, an on-line tool that is able to plan movie and dinner activities for a group of users, possibly splitting them in subgroups to optimally satisfy their preferences. We first model this problem and prove that it is NP-complete. We then use Constraint Programming (CP) or alternatively Simulated Annealing (SA) to solve it. Empirical results, obtained on real data for the city of Paris, show the feasibility and scalability of the approach even when hundred of users can select among hundreds of movies and thousand of restaurants.

It is worth noticing that even though, for the sake of clarity and concreteness, in this paper we focus on the above mentioned activities, our approach is completely general and our tool can be easily adapted to any problem which has the following features: 1) there is a group of users who have to perform a sequence of n activities; 2) each user can express some preferences on these activities; 3) the group can be divided in several sub-groups, each one performing a different activity at a given time frame; 4) temporal and spacial constraints can be added on the different activities; 5) the aim of the tool is to optimize the overall satisfaction of all the users involved in the activities.

Structure of the paper. In Section 2 we describe `NightSplitter` from the user perspective. In Section 3 we first formalize the problem solved by `NightSplitter` proving its NP-hardness while in Section 4 we present how CP and SA techniques are used to solve it. Section 5 presents the experiment results that validate the use of `NightSplitter`. Related work and conclusions are in Section 6 and 7 respectively.

2 NightSplitter

`NightSplitter`, the tool we have developed and that we present in this Section, is a web application for planning movie and restaurant activities in the city of Paris. It may be used by a group of users and it can split them in subgroups to optimally satisfy their preferences. The application uses real data for (currently) 13598 restaurants and 93 cinemas with 153 movies, which are stored in

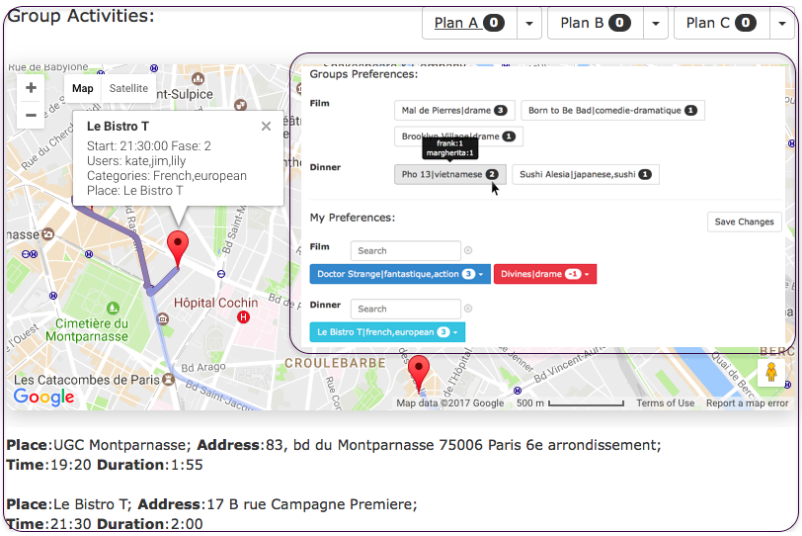


Fig. 1: NightSplitter Screenshot

a database and are constantly updated by a crawler embedded in the application. Using `NightSplitter`, an initial user dubbed *group initiator* can create a "group event" for a certain date. The group initiator is able to tune several parameters and constraints such as the number of possible subgroups, the size of subgroups, the total time window for performing the activities, the maximal time one is forced to wait between the activities. The group initiator can then invite other members to participate to the group by sharing a reference link. The invited member, by clicking on the link, is included automatically into the group and will be able to express his/her preferences, possibly inviting other persons to join the group.

As can be seen from Fig. 1 showing a screenshot of `NightSplitter`, by using some simple menus each user can express preferences on movies and restaurants in Paris. Social interaction among group members is possible, since each user can see the preferences of others and can instantly see the results of updating or modifying his/her own preferences. The main interface is divided in two parts: a dashboard for preferences and a digital map for showing the solutions. In the preference dashboard (right side of Fig. 1), users can input their preferred movie and restaurant names (or alternatively movie and cuisine categories). The introduction of this information is facilitated by an autocomplete function that suggest possible values. The expressed preference is represented by a tag with color, where the tag shows the name of the preference and the color indicates its scale: deep blue to signal a strong like, light blue for like, yellow for dislike, red for strong dislike, and gray for neutral. On the top of the dashboard, there is a summary of the group preferences, where in each tag, next to the activity name, there is an aggregated score. Each time a user enters or modifies a pref-

erence, the preference dashboard will be updated in real time and the system will start to compute a new solution.[4] The computation, as later detailed in Section 4, uses either a Constraint Programming or Simulated Annealing technique. The averages of the individual preferences and the public ratings of the selected activities are weighted and combined to form a unique evaluation metric to establish the quality of every solution (cf. Definition 6). The 3 solutions with highest aggregated preference are provided and displayed on-the-fly to the users, both in textual form and on the digital map. The text informs the user about their tentative scheduled activities while the map provides a global view of the subgroups activities with their cinema-restaurant paths. Given the different solution plans, group members have the option to like or dislike them by clicking "Plan A/B/C" as shown in the upper part of Fig. 1. Based on these votes the group initiator can finalize the decision and pick up the plan for the entire group.

The online version of `NightSplitter` is available at [29] [5].

3 NightSplit

In this section we formalize the definition of the optimization problem solved by `NightSplitter` and dubbed `NightSplit`. The key elements of `NightSplit` are the users and the activities that users can perform. We therefore assume the following finite disjoint sets: $\mathcal{U}$ for users range over by $u_1, u_2, \ldots$, $\mathcal{A}_M$ and $\mathcal{A}_R$ for the movie and restaurant activities respectively. We will denote with $\mathcal{A} = \mathcal{A}_M \cup \mathcal{A}_R$ a generic activity ranged over by $a_1, a_2, \ldots$.

Activities have properties such as a possible starting time or the location where they are performed. The planning problem therefore needs to consider two dimensions: time and space. As far as the time is concerned, for `NightSplit` we consider only a fixed time window assuming that we want to plan all the activities within a given time range. In particular, for simplicity we use a discrete notion of time dividing the time window in time slots of fixed duration. Similarly, we discretize also the space by dividing it into a finite number of different locations. The granularity of the time and the space can be arbitrarily improved by reducing the duration of the time slot or considering smaller locations. In the following we denote with $TIME = \{1, \ldots, T_{max}\}$ and $Loc = \{1, \ldots, Loc_{max}\}$ the time slots and the locations where T_{max} and Loc_{max} are the number of time slots and the number of locations. In our examples, we consider 5 min as the time slot unit. We can therefore define the general properties of an activity as follows.

Definition 1 (Activity Proprieties). *Given a set of activities $\mathcal{A}$ we denote with:*

- *`startTime` the total function $\mathcal{A} \to TIME$ that associates to an activity its starting time slot (i.e., when the movie starts or when the restaurant opens),*

[4] Currently preferences are visible to all the users. However, mechanisms to hide the individual preferences such as differential privacy [8] are under consideration.

[5] We are developing the tool for commercial use.

- ***endTime*** *the total function* $\mathcal{A} \to TIME$ *that associates to an activity its finishing time slot (i.e., when the movie ends or when the restaurant closes),*
- ***duration*** *the total function* $\mathcal{A} \to TIME$ *that associates to an activity the user's duration in time slots.*
- ***area*** *the total function* $\mathcal{A} \to Loc$ *that associates to an activity the location where it takes place.*
- ***publicRating*** *a complete function* $\mathcal{A} \to \mathbb{N}$ *that associates to an activity a possible rating.*[6] *Ratings are represented with natural numbers: the bigger the rating, the better the activity is considered.*

With a slight abuse of notation, given an activity a and a property p we denote with $a.p$ (rather than with p(a)) the value of the propriety p for activity a.

Example 1. A restaurant activity $a \in \mathcal{A}_r$ might be characterized by $a.\texttt{startTime} = 228$, meaning that the restaurant opens at 19:00 (assuming a time slot of 5 minutes 228 corresponds to 19), $a.\texttt{endTime} = 276$, meaning that the restaurant closes at 23:00, $a.\texttt{duration} = 18$ meaning that the dinner will last 90 minutes, $a.\texttt{area} = 5$ meaning that the location is identified with id 5, and $a.\texttt{publicRating = 3}$ meaning that the public rating is 3.

As far as preferences are concerned, based on findings such as those reported in [23], we avoid using a very refined scale and we allow only 5 values: from -2 indicating a strong dislike to a +2 indicating a strong preference, and 0 indicating a neutral opinion. Formally user preferences are defined as follows.

Definition 2 (Activity Preferences). *Given a set of users* $\mathcal{U}$ *and a set of activities* $\mathcal{A}$*, an activity preference is a total function* $\texttt{pref} : \mathcal{U} \times \mathcal{A} \to \{-2, -1, 0, 1, 2\}$.

Since the user has to move between different locations, to properly define a valid plan we need a metric that evaluates the distance between different activities. We are only interested in the time to go from one activity to another. Hence, we abstract from physical details such as GPS coordinates and means of transportation and we simply consider a distance metric between locations which is given in terms of times slots (needed to go from one location to the other).

Definition 3 (Distance metric). *Given a set of locations Loc and a set of time slots* $TIME = \{1, \ldots, T_{max}\}$ *a distance metric is a total function* $\texttt{dist} : Loc \times Loc \to TIME$.

We are now ready to define what is a plan: a simple association of activities to the users.

Definition 4 (Plan). *Let us consider a set of users* $\mathcal{U}$*, two sets of activities* $\mathcal{A}_M$ *and* $\mathcal{A}_R$ *and a set of time slots* $TIME$*. A plan is a total function* $\texttt{plan} : \mathcal{U} \to (\mathcal{A}_M \times TIME) \times (\mathcal{A}_R \times TIME)$ *that associates to a user a movie and restaurant activity with their beginning time slots.*

[6] Specifically, the rating value of activity ranges from 0 to 5, where 0 means "no rating information is given".

Example 2. A plan $\texttt{plan}(u) = ((a_1, 108), (a_2, 138))$ means that to the user u is assigned the activity a_1 that starts at 9:00 and the activity a_2 at 11:30.

Not all the plans are valid: For instance a plan may schedule two overlapping activities for a user. For this reason, we introduce the notion of plan validity that captures the constraints that a feasible plan must possess.

Definition 5 (Plan Validity). *Given a positive integer* $\mathit{maxGroupNum}$ *representing the maximal number of sub-groups allowed, a positive integer* $\mathit{minCardinality}$ *representing the minimal size of a group, and a positive integer* $\mathit{maxWait} \in TIME$ *representing the maximal waiting time between two activities, a plan* plan *is said valid iff:*

- *starting and ending time are satisfied. Formally, for each user* $u \in \mathcal{U}$*, if* $\mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))$ *then* $\mathit{startTime}(a_m) \leq t_m \leq \mathit{endTime}(a_m) - \mathit{duration}(a_m)$ *and* $\mathit{startTime}(a_r) \leq t_r \leq \mathit{endTime}(a_r) - \mathit{duration}(a_r)$*;*
- *activities do not overlap. Formally,* $\forall u \in \mathcal{U}$*, if* $\mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))$ *then* $t_r \geq t_m + \mathit{duration}(a_m) + \mathit{dist}(\mathit{area}(a_m), \mathit{area}(a_r))$*;*
- *activities are not too far apart. Formally,* $\forall u \in \mathcal{U}$*, if* $\mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))$ *then* $t_r \leq t_m + \mathit{duration}(a_m) + \mathit{maxWait}$*;*
- *the number of groups is limited by* $\mathit{maxGroupNum}$*. Formally,* $|\{(a_m, t_m) \mid \forall u \in \mathcal{U} \,.\, \mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))\}| \leq \mathit{maxGroupNum}$ *and* $|\{(a_r, t_r) \mid \forall u \in \mathcal{U} \,.\, \mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))\}| \leq \mathit{maxGroupNum}$ *;*
- *the cardinality of the group is bounded by* $\mathit{minCardinality}$*. Formally, for all activities* $a_m \in \mathcal{A}_m$*, and time slots* $t_m \in Time$ $|\{u \mid \forall u \in \mathcal{U} \,.\, \mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))\}|$ *is 0 or greater or equal than* $\mathit{minCardinality}$*. Similarly, for all activities* $a_r \in \mathcal{A}_R$*, and time slots* $t_r, \in Time$ $|\{u \mid \forall u \in \mathcal{U} \,.\, \mathit{plan}(u) = ((a_m, t_m), (a_r, t_r))|$ *is 0 or greater or equal than* $\mathit{minCardinality}$*.*

In order to simplify the presentation, given a plan $\texttt{plan}(u) = ((a_1, t_1), (a_2, t_2))$ in the following we will use $\texttt{plan}(u).a_m$ for denoting a_1, $\texttt{plan}(u).a_r$ for a_2, $\texttt{plan}(u).t_m$ for t_1, and $\texttt{plan}(u).t_r$ for t_2 (m stands for movie, r for restaurant).

We are now ready to define the `NightSplit` optimization problem. Intuitively, the `NightSplit` goal is to find a valid plan that optimizes the individual activity preferences and the public activity preferences. Different criteria may be used to combine these preferences. `NightSplit` allows a great flexibility combining all these objectives into one by summing them according to some weights.

Definition 6 (`NightSplit`).
Let η *be a real number* $\in [0, 1]$ *representing the weight associated to the individual activity preferences and the public preferences* [7]*. The* $\mathit{NightSplit}$ *problem is to find the valid plan* plan^* *that maximizes the following objective function.*

$$\mathit{obj}(\mathit{plan}) = \eta \cdot \mathit{sum}_{act}(\mathit{plan}) + (1 - \eta) \cdot \mathit{sum}_{pub}(\mathit{plan}) \tag{1}$$

[7] Public preferences are useful to break the ties when users have very general individual preferences (e.g., I like all the movies)

where $\mathtt{sum}_{act}$ *and* $\mathtt{sum}_{pub}$ *are the sum of the individual activities preferences and public preferences as define below:*

$$\mathtt{sum}_{act}(\mathtt{plan}) = \sum_{u \in \mathcal{U}} (\mathtt{pref}(u, \mathtt{plan}(u).a_m) + \mathtt{pref}(u, \mathtt{plan}(u).a_r)) \tag{2}$$

$$\mathtt{sum}_{pub}(\mathtt{plan}) = \sum_{u \in \mathcal{U}} (\mathtt{plan}(u).a_m.\mathtt{publicRating} + \quad \mathtt{plan}(u).a_r.\mathtt{publicRating}) \tag{3}$$

As can be expected, even tough this formulation is rather simple, `NightSplit` is an NP-hard problem.

Theorem 1 (NP-hardness). *The* `NightSplit` *is NP-hard.*

Proof. To prove hardness, we reduce the NP-complete problem Perfect Expected Component Sum (PECS) [4] to the decision version of `NightSplit`, i.e., the problem to find whether there exists a valid plan such that the objective function `obj` is greater or equal than a given value. [8] An instance of PECS consists of a collection V of m-dimensional boolean vectors, i.e., $V \subset \{0,1\}^m$ and a number k. The problem is to determine whether there exists a disjoint partition of V into k subsets $V_1, \ldots, V_k$ such that $\sum_{i=1}^{k} \max_{1 \leq j \leq m} (\sum_{\bar{v} \in V_i} \bar{v}|j|) = |V|$.

Given an instance of PECS we map every vector $\bar{v}_i \in V$ as a user u_i having some preferences over m different movies. The intuition behind the hardness proof is to exploit the planning of the movie activities to find a solution for PECS. We assume that there is only one location, that the m movie activities start at the time slot 0 and end at time slot 1 with duration 1. Similarly, we assume that there are m different restaurant activities that start at time slot 1 and end at time slot 2 with duration 1. We set `maxGroupNum` to k, `minCardinality` to 1, `maxWait` to 1, and we assume that the function `dist` is the constant function 0. In this way all the movie activities are compatible with the restaurant activities and all the possible plans that have a maximal number of k groups are valid. We set the preferences of the movie activities to reflect the values of the vector $\bar{v}$. Formally, for all $1 \leq i \leq |V|$ and $1 \leq j \leq m$ we define $\mathtt{pref}(u_i, a_j) = \bar{v}|j|$. We set to 0 instead the preferences for all the restaurant. We set the weight of the user preferences η to 1 while we discard the public preferences with $1 - \eta = 0$.

Based on the definition of `NightSplit`, it is easy to see that $\sum_{i=1}^{k} \max_{1 \leq j \leq m} (\sum_{\bar{v} \in V_i} \bar{v}|j|) = |V|$ iff the `obj` of the `NightSplit` problem is equal to $|V|$. The partition induced on the users performed by `NightSplit` corresponds to the partition of V into the k set of vectors $V_1, \ldots, V_k$. □

[8] The decision version of the problem requires the "greater or equal" operator. Similar to the theorem presented in [4], our theorem holds because the sum of the preferences is never greater than V.

3.1 Useful Extensions

While `NightSplit` is already NP-hard, there are some useful extensions of it that do not alter its complexity class and its nature. In the following we just comment on some of them that are considered in the online `NightSplitter`. For space reasons they are not formally defined here, however their definition is straightforward.

First observe that the notion of a valid plan can be further restricted considering additional constraints. For example, it may be useful to allow users to indicate that they are not available before or after a given time. Moreover, the minimal number of people required to form a group or the number of groups can vary depending on the activity (e.g., it may be the case that for going to the movie we accept to split the group in two while to eat in a restaurant we do not allow any split). Other useful extensions concern the definition of different kinds of user preferences. For instance, usually users like to hang out in certain locations and they want to minimize the traveling time between the activities, minimize the waiting time, start the activities as soon as possible, etc. All these preferences may be considered by adding further terms to the objective function that we optimize in `NightSplit`, possibly reducing its weight by an appropriate parameter. `NightSplitter` has been designed to be easily extensible and take into account new sources of user preferences or constraints. For instance, the preferences over some areas can can be easily defined in the profile menu of the user and then taken into account when generating the plans.

Finally, we could also relax the limit of two activities, considered in this paper, and we could extend our system to applications where more activities can be performed in sequence, especially in the tourism industry, following, e.g., [25,18].

4 Solution Approaches

To solve the `NightSplit` problem we propose two different approaches. The first one relies on Constraint Programming (CP) and allows us, in principle, to obtain the optimal solution. The second approach uses Simulated Annealing (SA), a probabilistic local search procedure which, under certain conditions for its parameters, is known to find the optimal solution with a probability approaching one. In this section we briefly describe the CP and SA approaches, while we defer to Section 5 for their comparison.

4.1 NightSplit and Constraint Programming

Constraint Programming (CP) [24,26] is a widely adopted approach for solving NP-hard problems. The CP paradigm enables to express complex relations in form of constraints to be satisfied. In particular a Constraint Satisfaction Problem (CSP) $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ consists of a finite set of variables $\mathcal{X}$, each of which associated with a domain $D_x \in \mathcal{D}$ of possible values that it could take, and a set of constraints $\mathcal{C}$ that defines all the admissible assignments of values to the

variables [19]. Given a CSP the goal is normally to find a solution, i.e, an assignment to the variables that satisfies all the constraints of the problem. When an objective function needs to be minimized or maximized we deal instead with a Constraint Optimization Problems (COPs), i.e., a generalized CSP where the goal is not only to find a solution but among all possible solutions the one that maximizes or minimizes the objective function.

Clearly the `NightSplit` problem can be seen as a COP. For every user u we have introduced:

- a variable M_u representing the selection of the movie activity. The domain of this value is the finite domain of all the possible movie activities;
- a variable R_u representing the selection of the restaurant. The domain of this value is the finite domain of all the possible restaurant activities;
- two variables $S_{u,1}, S_{u,2}$ representing the beginning of the activities. The domain of these variables is the finite set of the possible time slots;
- two variables $G_{u,1}, G_{u,2}$ representing the subgroup to which user u belongs (for the first and second activity respectively). The domain of these variables depend on the maximal number of groups allowed for activity.

With these variables it is possible to state all the constraints as listed in Definition 5. For instance, the first constraint bounding the starting time of the activities might be expressed by stating that $\texttt{movie\_start}[M_u] \leq S_{u,1}$ where `movie_start` is the array storing the movies starting time. This constraint is simply a disequality between two expressions: the first retrieves the concrete value from an array while the second is the variable $S_{u,1}$. Note that CP solvers can employ efficient techniques to handle this kind of equalities or disequalities (global constraints). Moreover, for this particular case, the constraint setting x as the value taken by the y-th value of the array is known as element constraint [24], which is often supported by constraint solvers that adopt ad-hoc propagation algorithms to speed up the search of solutions.

To model all the constraints we used MiniZinc [21], which is the de-facto language to define CSPs and COPs and is supported by a huge variety of constraint solvers. Since the majority of the solvers does not support real variables, we restrict the use of the preference weights η to rational numbers only. A detailed explanation of the MiniZinc model and all the constraints defined is outside the scope of this paper. For more information we invite the reader to consult [14].

Remark 1. Beside CP, we have also tried to encode the `NightSplit` to exploit Satisfiability-Modulo-Theories (SMT) solvers. SMT solving extends and improves upon SAT solving by introducing the possibility of stating constraints in some expressive theories, e.g., arithmetic or bit-vector expressions. While all the constraints of `NightSplit` can be encoded in SMT, we were not able to provide an encoding linear w.r.t. the number of activity locations. Indeed, differently to what happens in CP where the *element* constraint can be used [24], in the SMT case the encoding of the traveling time between two activities requires the introduction of a quadratic number of constraints w.r.t. the number of locations. Based on our test, since we had more than 300 locations, the addition of

these quadratic number of constraints hindered the use of SMT solvers. For this reason, in Section 5, we will compare only the performances of the CP and SA approaches.

4.2 NightSplit and Simulated Annealing

Simulated Annealing (SA) [1] is a local search technique inspired by the annealing process in metallurgy. SA has been widely used for approximating the global optimum of a given function. Given an initial solution, random moves are made to produce new potential solutions. A new solution that improves the previous one is (usually) always accepted. Solutions that worsen the current solution are instead accepted with a probability that, like the temperature in the annealing process, is gradually decreasing. Accepting worse solutions is a fundamental property because it allows for a more extensive search for the optimal solution, possibly avoiding getting stuck in local optima.

Contrary to the CP technique described before, SA can not guarantee that the final solution obtained is optimal. However, for discrete and large search spaces, SA scales better and could produce (sub)optimal solution very quickly.

Among all the different implementations of SA available we rely on the reimplementaiton in PHP of the python SA module [22]. After some manual tuning, we have fixed the parameters to control the decreasing of the temperature and the number of iterations (50000). The temperature exponentially decreases as the algorithm progresses. As customary, a move causing a decrease in state energy (i.e., an improvement of the `NightSplit` objective function) was always accepted. Moves instead increasing the state energy (i.e., a worse solution) but within the bounds of the temperature are also accepted.

The initial solution is obtained by randomly generating the assignments from users to activities. To obtain instead a valid plan from a current solution we proceed as follows: (i) we randomly select movie activity assignments or restaurants activity assignments and modify them; (ii) we randomly select a subset of users U; (iii) we assign a new activity a to the selected users in U. This activity is randomly chosen among all the activities for which the aggregated preference of the U users is positive. Intuitively, this avoids selecting an activity that no user in U wants to perform; (iv) if the assigned activity is not compatible with other existing ones (e.g., if for user u we select a movie activity a that overlaps with his/her restaurant activity) we delete these activities; (v) for every user u that has no activity assigned we look at the activities assigned to other users, check if any of them is compatible with the updated activity and if so we assign this activity to the user u assuming that this does not violate the group constraints.

5 Empirical Experiment

In this section we describe the experiments performed in order to validate the scalability of `NightSplitter` and we discuss the results.

Activity Type	Activities	Users	Avg. pref
Movies	1950	5300	6
Restaurants	17069	8000	2

Table 1: Summary Statistics of the Dataset

We have considered for the experiments real data from the city of Paris: The movies information - for 93 cinemas and currently 153 different movies (with 1950 projections a day) - is retrieved from Allociné [2,10], restaurant data - for 13598 restaurants - from Tripadvisor [30]. OpenStreetMap and GoogleMaps were also used to identified 317 positions of metro stations: for each activity we considered its nearest metro station as its location. The data related to the preferences was collected from Movielens [12] and Yelp [31]. These datasets, originally defined for activities in the U.S., were converted for Paris activities. This was done by mapping the names of the Paris activities to the activity existing in the preference dataset while preserving the activity category and the public rating. After that, we randomly sampled 8,000 users for the restaurant activity and 5.300 users for the movies activity to use their individual preferences for the experiments. The statistics related to the activities and preference data are summarized in Table 1 where the last column indicates the average preferences of the users. Note that if a restaurant was open for two separate intervals (e.g., from 11 to 15 and from 19 to 23) this was captured by considering two separate activities.

Since the goal is to provide a responsive tool, for the experiments we fixed a timeout of 60 seconds taking the best solution found by the tested approach within this time frame. For each testing scenario we repeated the experiment 30 times. For every experiment we match the chosen number of user with random user from the dataset using their preferences. We allow the subgroups to be formed by at least 2 people, the time slot unit to be 5 minutes assuming that the duration for a dinner/lunch is 90 minutes. The experiments were run on an Ubuntu Intel Core 3.30GHz machine with 8 GB of RAM.

We compared the performance of three different state-of-the-art CP solvers, namely Chuffed [6], Or-Tools [11], and HCSP [13], [9] and the SA method described in the previous section.

We first compare the three different CP solvers for different number of users, assuming to have only 2 subgroups and not taking into account the public ratings (i.e., $\eta = 1$). Fig. 2 shows the average times needed by the solvers to find the optimal value by varying the number of users, where the filled icons mean that the solver has proven the optimality of the solution for all the 30 repeated tests. Chuffed has always computed the optimal solution for values up to 9 users and it is the fastest among the three solvers. The Or-Tools cannot find the optimal

[9] We selected these solvers based on the recent results of the MiniZinc Challenge 2016 [27]. In particular Or-Tools won a golden medal in the Fixed category and HCSP won a golden medal in Free and Parallel category. Chuffed was the second best solver of the entire Challenge after LCG-Glucose-free which is not publicly available. We would remark also that our problem instances have been submitted to the incoming MiniZinc Challenge 2017 [28].

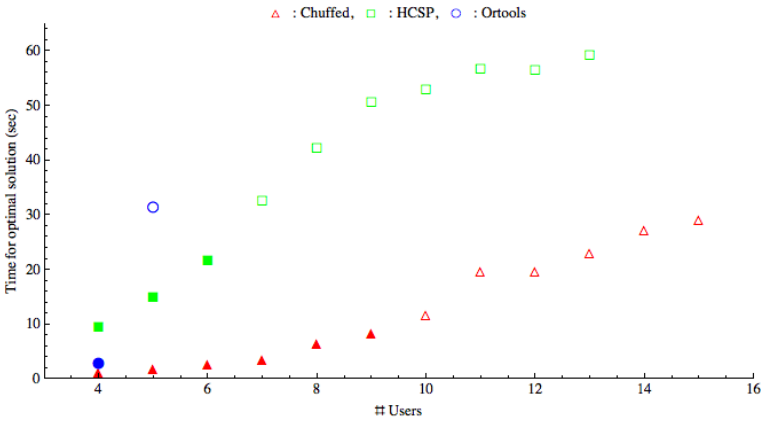


Fig. 2: CP Solvers comparison.

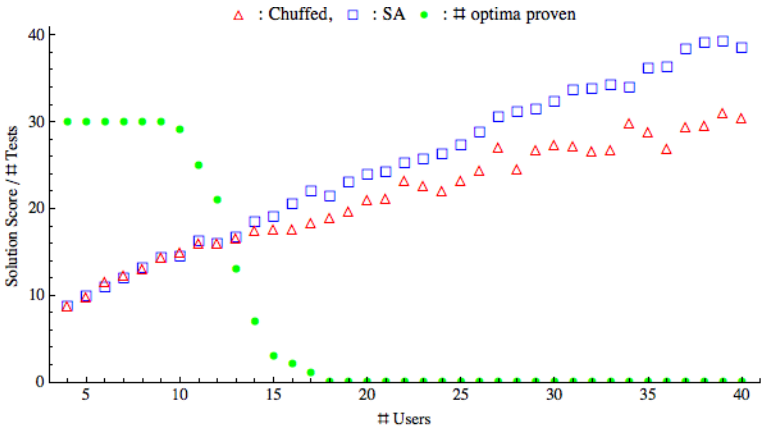


Fig. 3: CP vs SA comparison.

solution within the timeout for more than 5 users, while the HCSP solver performs slightly better than Or-Tools and occasionally it is still capable to prove optimal solution for up to 13 users. Similar results are obtained when increasing the number of subgroups or when public ratings are taken into account by lowering the value of the η parameter. Since Chuffed outperforms the other solvers in our application, in the following we show only the performance of this solver for the comparison with SA approach.

We compare the performance of Chuffed and SA in terms of quality of the solution for a number of users ranging between 4 and 40, assuming 2 subgroups could be formed, and the weight associated to the individual preference η to be 1. Fig. 3 and 4 depicts respectively the average solution score and the average time needed to find the best solution for the 30 repeated tests (the green dot in Fig. 3 representing the number of tests such that CP proves solution optimality). The plots show that for a limited number of users SA is competitive with Chuffed,

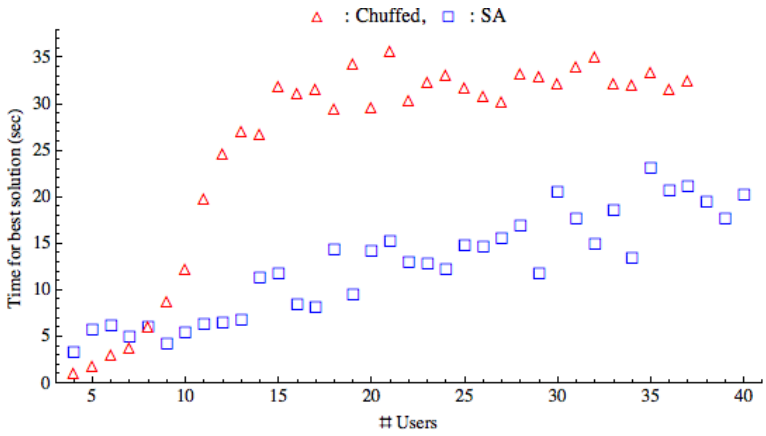
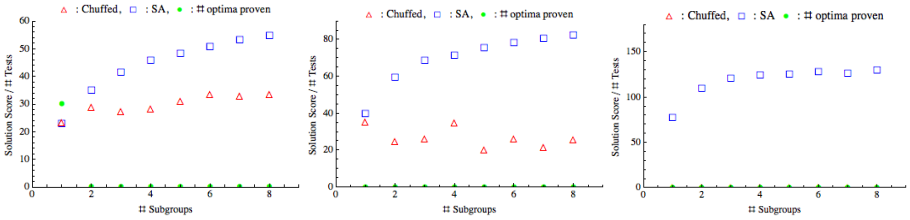


Fig. 4: Time to find the best solution.

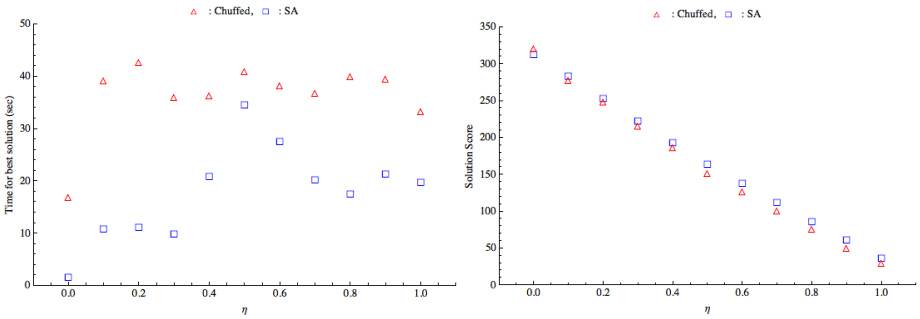


(a) 32 users (b) 64 users (c) 128 users

Fig. 5: Comparison of CP and SA varying the number of subgroups.

while for more than 15 users SA is definitely better. The advantage of the CP solution is that for less than 10 users the solutions are proven optimal while some SA solutions were suboptimal. From the plot it is however possible to see that the number of solutions that could be proven optimal in less than 60 second decreases at the increase of the number of users. With more than 20 users no solution was proven optimal. It is clearly visible that Chuffed is better only for a limited number of users while the SA is often able to find the best solution within the first 15 seconds.

We then compare the two approaches by varying the number of possible subgroups from 1 to 8. In Fig. 5 we present the plots obtained considering 32, 64, and 128 users. From the plots it can be seen that the CP technique is only suitable with few users and when no more than 2 subgroups can be formed. When the number of users increases or more than 2 groups can be formed the solutions provided by the CP solver within 60 seconds are worse than the ones produced by the SA. In our biggest scenario, considering 128 users, the SA is the only viable choice because unfortunately the CP solver is not even able to provide a single solution (hence the lack of points for Chuffed in Fig. 5c). We conduct experiments also varying the weights used to aggregate the individual and public preferences. In these cases there are no significant changes, except that



(a) Time to find the best solution. (b) Average score of best solutions.

Fig. 6: Comparison of CP and SA varying η.

the final score increases. Fig. 6 shows for instance the performances of Chuffed and SA while varying the parameter η considering 32 users and 2 subgroups. In particular, Fig. 6a presents the average time when the best value is found while Fig. 6b presents the average score found after 60 seconds. As long as the user's preferences are accounted for (i.e., $\eta \neq 0$), it is immediately visible that with this amount of users the SA approach is better than Chuffed since SA is able to find better values in a short amount of time and Chuffed is not able to prove the optimality of the solutions within 60 seconds.

Summarizing, we may conclude that when considering two subgroups and few users the CP approach may be useful and even prove the optimality of the solution. For more subgroups and more users the SA approach is better. For those experiments where the optimality of the solutions was proven, the SA approach was able to propose competitive solutions. We conjecture that this holds also for big instances where we were not being able to prove the optima.

6 Related work

The literature on recommender or planning systems is very large and we omit all the references to works which consider the case of a single user only, with the exception of [25], which uses CSP techniques for building a tourist recommendation and planning application. Concerning group recommender systems, [5] provide a survey on several existing approaches while [9] presents a recommender system for tourism based on the tastes of the users, their demographic classification and the places they have visited in former trips. More recently, the idea of group splitting has appeared in some papers. Notably [4] proposes an approach for forming groups of users in order to maximize satisfaction. The work [18] introduces the problem of group tour recommendation which includes the problem of forming tour groups whose members have similar interests. Differently from our case, all the above mentioned papers consider groups or sub-groups as fixed entities, which once are created cannot be modified. With our approach, instead, for each activity we have a different group formation, that is, we can have two users who are in the same group for the first activity (the movie) and are in

different groups for the second one (the dinner). Moreover, the above papers focus on the theoretical aspects rather than presenting a tool.

There exist also several works which address the problem of group preference modeling and the definition of an appropriate notion of "group satisfaction" [20,16]. In general these are difficult tasks, since it is hard to find a definition which takes into account all the various aspects involved in the group dynamics. An interesting approach is presented in [3], where the notion of disagreement between group members is formally defined and, on its basis, a consensus function is introduced in order to formally define a satisfactory semantics for group recommendation. In some cases, users preferences depend on the contextual information in a dynamic domain, thus making even more difficult to make recommendation for groups. Recently Context-Aware Recommender Systems [15] have been proposed in order to address this issue. All the above mentioned approaches to the modeling of preferences, while interesting and relevant, are somehow orthogonal to the problem that we are considering in our paper. Indeed, we could easily change the preference model without major changes in our tool.

To conclude we would like to mention also the works conducted in [7,17,25] which present recommendation and planning systems targeting a single user only but are interesting for us since they consider models of generating itineraries (for touristic applications) which could be integrated with our tool.

7 Conclusions and future work

We have presented `NightSplitter`, an on-line tool that is able to plan movie and dinner activities for a group of users, possibly splitting them in subgroups to optimally satisfy their preferences. The tool is based on a formal model and two different technologies - Constraint Programming and Simulate Annealing - which can be easily adapted to other applications. The tests we have conducted show that our tool can be effectively used on real data for the city of Paris, with thousands of activities and hundred of users. The comparison between CP and the simulated annealing approach show that the latter can scale up to consider larger number of users, making our approach feasible also for quite different social applications.

We are now extending our work along several directions: First, we are considering a greater number of different activities and we are adding some more features such as, e.g., the selection of a preferred limited area for the activities (this is done by selecting an area on the map). Second, the recommendation semantics adopted in our model is aggregated preference: we are now exploring different notions of group recommendation semantics such as least misery, most pleasure, Borda count, etc. [20]. In particular we would like to see whether the semantics proposed in [4] with the related algorithms could improve our approach. Third, we would like to investigate techniques for group definition using social factors and group dynamics as those suggested in [16]. Fourth, we would like to explore possible improvements for the CP approach by using, e.g., linearizion of the constraints, column generation methods, or the use of presolve.

References

1. Aarts, E.H., Korst, J.H.: Simulated annealing. ISSUES 1, 16 (1988)
2. AlloCiné: Allociné website (2016), available at `http://www.allocine.fr`
3. Amer-Yahia, S., Roy, S.B., Chawlat, A., Das, G., Yu, C.: Group recommendation: Semantics and efficiency. Proceedings of the VLDB Endowment 2(1), 754–765 (2009)
4. Basu Roy, S., Lakshmanan, L.V., Liu, R.: From group recommendations to group formation. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. pp. 1603–1616. ACM (2015)
5. Boratto, L., Carta, S.: State-of-the-art in group recommendation and new approaches for automatic identification of groups. In: Information retrieval and mining in distributed environments, pp. 1–20. Springer (2010)
6. Chu, G., de la Banda, M.G., Mears, C., Stuckey, P.J.: Symmetries and lazy clause generation. In: Proceedings of the 16th international conference on principles and practice of constraint programming (CP'10) Doctoral programme. pp. 43–48 (2010)
7. Di Bitonto, P., Di Tria, F., Laterza, M., Roselli, T., Rossano, V., Tangorra, F.: A model for generating tourist itineraries. In: 2010 10th International Conference on Intelligent Systems Design and Applications. pp. 971–976. IEEE (2010)
8. Dwork, C.: Differential privacy: A survey of results. In: TAMC. LNCS, vol. 4978, pp. 1–19. Springer (2008)
9. Garcia, I., Sebastia, L., Onaindia, E.: On the design of individual and group recommender systems for tourism. Expert systems with applications 38(6), 7683–7692 (2011)
10. Gauvin, E.: Allocine helper (2016), available at `https://github.com/etienne-gauvin/api-allocine-helper`
11. Google: Google or-tools (2016), available at `https://developers.google.com/optimization/`
12. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5(4), 19 (2016)
13. Ivrii, A., Ryvchin, V., Strichman, O.: Mining backbone literals in incremental sat. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 88–103. Springer (2015)
14. Jacopo Mauro, T.L.: Minizinc model (2017), available at `http://cs.unibo.it/t.liu/nightsplitter/mzn.html`
15. Khoshkangini, R., Pini, M.S., Rossi, F.: A self-adaptive context-aware group recommender system. In: AI* IA 2016 Advances in Artificial Intelligence, pp. 250–265. Springer (2016)
16. Kompan, M., Bielikova, M.: Group recommendations: Survey and perspectives. Computing and Informatics 33(2), 446–476 (2014)
17. Le Berre, D., Marquis, P., Roussel, S.: Planning personalised museum visits. In: ICAPS (2013)
18. Lim, K.H., Chan, J., Leckie, C., Karunasekera, S.: Towards next generation touring: Personalized group tours (2016)
19. Mackworth, A.K.: Consistency in Networks of Relations. Artif. Intell. 8(1), 99–118 (1977)
20. Masthoff, J.: Group recommender systems: Combining individual models. In: Recommender systems handbook, pp. 677–702. Springer (2011)
21. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International Conference on Principles and Practice of Constraint Programming. pp. 529–543. Springer (2007)

22. Perry, M.: Python module for simulated annealing (2017), available at https://github.com/perrygeo/simanneal
23. Preston, C.C., Colman, A.M.: Optimal number of response categories in rating scales: reliability, validity, discriminating power, and respondent preferences. Acta psychologica 104(1), 1–15 (2000)
24. Rossi, F., Van Beek, P., Walsh, T.: Handbook of constraint programming. Elsevier (2006)
25. Sebastia, L., Garcia, I., Onaindia, E., Guzman, C.: *E-Tourism*: a tourist recommendation and planning application. International Journal on Artificial Intelligence Tools 18(5), 717–738 (2009), http://dx.doi.org/10.1142/S0218213009000378
26. Smith, B.M.: Modelling for constraint programming. Lecture notes for the first international summer school on constraint programming (2005)
27. Team, M.: Minizinc challenge 2016 (2016), http://www.minizinc.org/challenge2016/challenge.html
28. Team, M.: Minizinc challenge 2017 (2017), http://www.minizinc.org/challenge2017/challenge.html
29. Tong Liu, J.M.: Nightsplitter (2017), available at http://cs.unibo.it/t.liu/nightsplitter
30. TripAdvisor: Tripadvisor website (2016), available at https://www.tripadvisor.com
31. Yelp: Yelp dataset challenge (2016), available at http://yelp.com/dataset_challenge/