SNeT: computer-assisted SuperNovae Tracking

by

Di Bao

B.E. in Software Engineering, UESTC, 2011

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2013

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This thesis was presented

by

Di Bao

It was defended on

November 25, 2013

and approved by

Panos K. Chrysanthis, Professor

Alexandros Labrinidis, Associate Professor

Michael Wood-Vasey, Assistant Professor

Thesis Director: Panos K. Chrysanthis, Professor

Alexandros Labrinidis, Associate Professor

Copyright © by Di Bao

2013

SNeT: computer-assisted SuperNovae Tracking

Di Bao, M.S.

University of Pittsburgh, 2013

In astronomy, supernovae are stellar explosions whose observation can help shed light on the star formation process and provide reference points for cosmological distances. Supernovae are detected at different phases of their lifecycle and their observation is further complicated by time and resource constraints. Although there exists automated supernovae detection pipelines, follow-up observations by individual researchers are handled manually, both in terms of keeping a list of interesting supernovae worth observing and also planning out the exact schedule for observations, given telescope access and temporal constraints.

This thesis designs and develops the SNeT (computer-assisted SuperNovae Tracking) system, as a tool to help astronomers collect supernovae data, manage their lists of interest and observation plans, and most importantly, generate good observation plans automatically, that can later be further adapted. Specifically, SNeT takes a list of supernovae, their associated temporal constraints, and user preferences, and it generates a plan that satisfies the constraints and preferences, maximizes data acquisition, while minimizing time and resource usage. In addition, the user can interact with the system and give feedback on the generated plans in order to customize SNeT's planning behavior via its self-tuning. The SNeT prototype system is currently evaluated by supernovae researchers from the Department of Physics and Astronomy of the University of Pittsburgh.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ALGORITHMS

PREFACE

For Xuanwei. For my parents. For Panos and Alex.

1.0 INTRODUCTION

1.1 MOTIVATION

In astronomy, a supernova [1] is a stellar explosion that is more energetic than a nova, which is a nuclear explosion in a white dwarf star. Supernovae are extremely luminous and cause a burst of radiation that often briefly outshines an entire galaxy (Figure 1.1.1), before fading from view over several weeks or months. Supernovae can provide important information on cosmological distances, and the understanding of the formation of stars. Supernova observation is the foundation and basic prerequisite for further studies of supernova. The history of supernova observation can be backtracked to ancient China (the earliest recorded supernova, SN 185, was viewed by Chinese astronomers in 185 AD), while the field of supernova discovery has extended to galaxies beyond the Milky Way quickly, after the development of the telescope.

Nowadays, as more supernovae are discovered and more astronomers put their scientific interests in this field, individual astronomers are faced with the challenge of keeping track and planning out the exact schedule of observations of supernovae of their interest. The main concerns are time and resource constraints. The time constraints are primarily due to the physical properties of supernovae. Supernovae are transient events whose lifetime is marked by a peak brightness after which their brightness decreases over time (according to their type as shown in Figure 1.1.2). When supernovae are discovered, they are already in progress and any follow-up

observations must be carefully planned for these observations to be productive. The planning of observations by individual astronomers is further complicated by the observation equipment availability (how many Hubble Space Telescopes are there?).

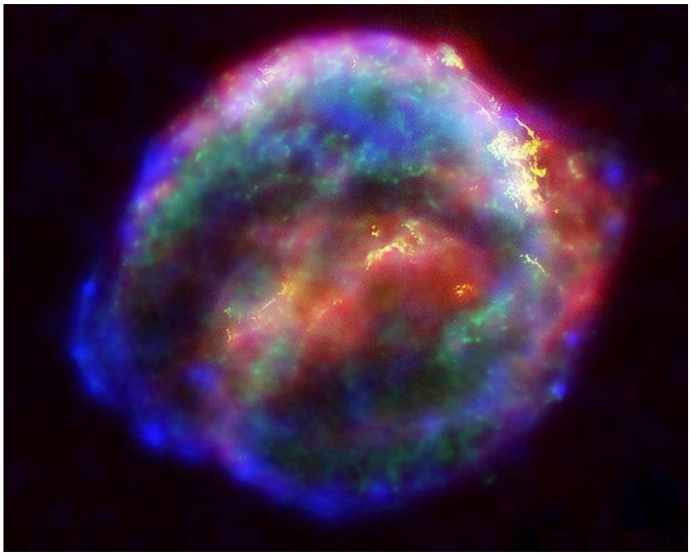


Figure 1.1.1 Supernova in explosion (Multiwavelength X-ray, infrared, and optical compilation image of Kepler's supernova remnant, SN 1604) [2]

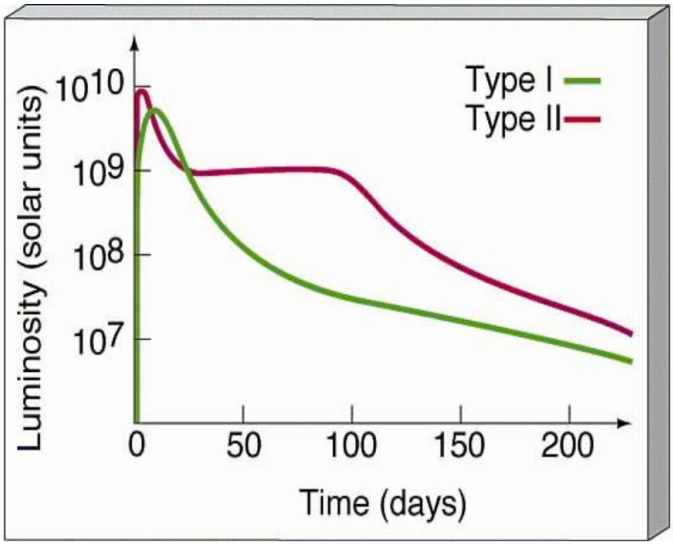


Figure 1.1.2 Typical light curve of supernova [3]

Basically, although there exists automated supernovae detection pipelines, follow-up observations by individual astronomers are handled manually, which clearly inefficient and time consuming, given telescope access and supernovae temporal constraints. Similarly, the astronomers manually collect, analyze, aggregate and cross-match supernova data from multiple sources (such as *Skyalert* [4], *ATel* [5], *CBAT* [6]) in order to identify interesting supernovae worth observing. The volume of messages/reports generated in near real-time by these sources makes this identification task a very difficult and even more time consuming.

This thesis is motivated by the lack of a supernova observation information system which integrates appropriate tools that help and simplify astronomers' tasks to collect supernovae data, manage their lists of interests and ongoing observation with associated annotations and hypotheses and most importantly, to generate good observations plans automatically.

1.2 PROBLEM STATEMENT

Current researchers of supernovae are faced with three key problems, which are described below.

Problem 1: Currently astronomers lack a consistent way to access supernova data published by different organizations (sources). Many organizations maintain their own services to collect and distribute supernova related information in the form of a message or a report. Unfortunately, different organizations use different formats (such XML, semi-structured XML and HTML) for their messages and reports. Furthermore, although these messages/reports from different sources could describe the same supernova object, they may use different supernova naming convention, and provide different supplemental data (such as *redshift*, *magnitude*, etc.).

Ideally, astronomers should be provided with an interface to access all the information about a certain supernova published in all the related messages/reports.

Problem 2: Currently astronomers lack a set of tools that helps them to create and maintain their target list of supernovae, save and retrieve their experiment configurations, record their observation plans as well as record and share their annotations on observed supernova objects.

Problem 3: Currently astronomers lack a tool to automatically generate high quality observation plans under limited time and resource constraints. The quality of a plan (for a specific set of supernovae and specific dates) is defined in terms of maximum data acquisition (i.e., number of successful supernovae observations), normalized by the value or importance of each supernova. The value of a supernova may vary from one astronomer to another and may also vary from one experiment to another for the same astronomer. The number of successful supernovae observations is determined by the temporal constraints associated with each supernova and the spatial proximity among supernovae, which define whether or not their observation window overlap during the same night.

1.3 APPROACH

We propose to address the above three key problems by developing a system consisting of four components: *Supernova Data Aggregator* (SDA) which addresses Problem 1, *Supernova Tracking Management System* (STMS) which addresses Problem 2 and *Basic Observation Scheduling* (BOS) and *Advanced Observation Scheduling* (AOS) both of which address Problem 3.

1.3.1 Supernova Data Aggregator (SDA)

SDA builds up a supernova data repository by integrating supernova data from different sources. It downloads supernova related message/report entries from multiple sources periodically, and adds any new released entries into a database. It further scans and parses those entries to extract information about supernova objects, while attempting to match objects from different sources.

1.3.2 Supernova Tracking Management System (STMS)

STMS provides the following functionalities to astronomers: search for interesting supernova records, build and maintain target lists of supernovae for observation, create and update particular experiment configurations for planning observations, record and display observation plans. STMS implements these functionalities by interacting with SDA, BOS and AOS.

1.3.3 Basic Observation Scheduling (BOS)

BOS provides a systematic way to automatically generate an observation plan for a given set of supernovae over a set of specific dates. Basically, it combines all the factors that could potentially affect the scheduling of observations and quantifies their effects, with a heuristic function, which is used in making scheduling decisions. At the global level, BOS generates candidates list of supernovae to be scheduled for each night and at the local level, BOS generates a feasible schedule of supernova observations.

1.3.4 Advanced Observation Scheduling (AOS)

AOS enhances observation plan generation with two additional features. The first feature defines the *shape* of the entire observation plan (i.e., across all dates) by reserving specific observation windows per date when generating the observation plan for a given target list. The goal of this reservation is to enable dynamic expansion of the observation target list during the observation plan execution. In this way, new and potentially very interesting supernovae, which are discovered after the generation of the observation plan, could be potentially included in the plan. Furthermore, failed observations during a specific night would potentially need to be accommodated at a later date. That is, this feature allows for the flexibility of "planning before it happens".

The second feature provides semi-supervised learning to enhance the quality of an observation plan. As mentioned above, the quality of an observation plan is (astronomer) user-specific. Briefly, the system maintains a training set of users' feedbacks on generated plans and learns from them to adjust the scheduler's behavior to meet the user preference. As a result, for the same target list, different users could expect different observation plans, according to how they "train" the system.

1.4 CONTRIBUTIONS

The main contributions of this thesis can be summarized as following:

- Designed and implemented the algorithms of the Supernova Data Aggregator (SDA) and built the Supernova Data Repository containing all the published data about supernovae since March 2013.
- Designed and developed a Supernova Tracking Management System (STMS) which allows the creation of observation target lists from the supernovae data in the Supernova Data Repository and the planning of observation experiments using a Supernova Plan Generator.
- Designed and implemented a Supernova Observation Plan Generator (SOPG) consisting of a Basic Observation Scheduling (BOS) module and Advanced Observation Scheduling (AOS) module. BOS is a suite of a baseline (exhaustive search) scheduler and three heuristic scheduling algorithms. Specifically, our proposed heuristics algorithms are:
  1. *Greedy Beam Search*, based on greedy approximation approach.
  2. *Iterative Dynamic Programming*, based on dynamic programming technique.
  3. *Random Restart Hill Climbing*, based on local search solving optimization problems.

  Our preliminary results have shown that all three algorithms are (1) scalable and (2) successfully balance the speed for generating a plan and the quality of the generated plan as compared to the baseline.

- Developed three capacity reserving strategies for shaping an observation plan as part of the AOS module:
    1. *Normal*, no space reservation (i.e., no built-in spare observation capacity). During planning, the capacity of all specified dates can be used and observations are scheduled as early as possible. Whatever capacity is not used, it is considered reserved capacity for future use and typically appears in the latest dates of the pre-specified set of dates.
    2. *Evenly*, reserves the same amount of space observation capacity across all pre-specified dates assuming that all observations can be uniformly scheduled on all dates. Whatever capacity is not reserved is used during the planning.
    3. *Inversed*, the opposite of "Normal". During planning, observations are scheduled at the latest possible time if they still do not affect the task's schedulability. The residual capacity is reserved for future use and typically appears at the earliest dates of the pre-specified set of dates.
- Developed semi-supervised learning feature based on logistic regression to enhance the quality of an observation plan, which is also integrated into Advanced Observation Scheduling (AOS) module.

All the above contributions were developed based on the requirements of and the feedback of supernovae researchers of the Department of Physics and Astronomy at the University of Pittsburgh and led to the SNeT (computer-assisted SuperNovae Tracking) prototype system (Figure 1.4.1). SNeT is accessible from the AstroShelf platform [7, 8], i.e., it is part of AstroShelf's web-based user interface and is currently evaluated by our supernovae research collaborators.

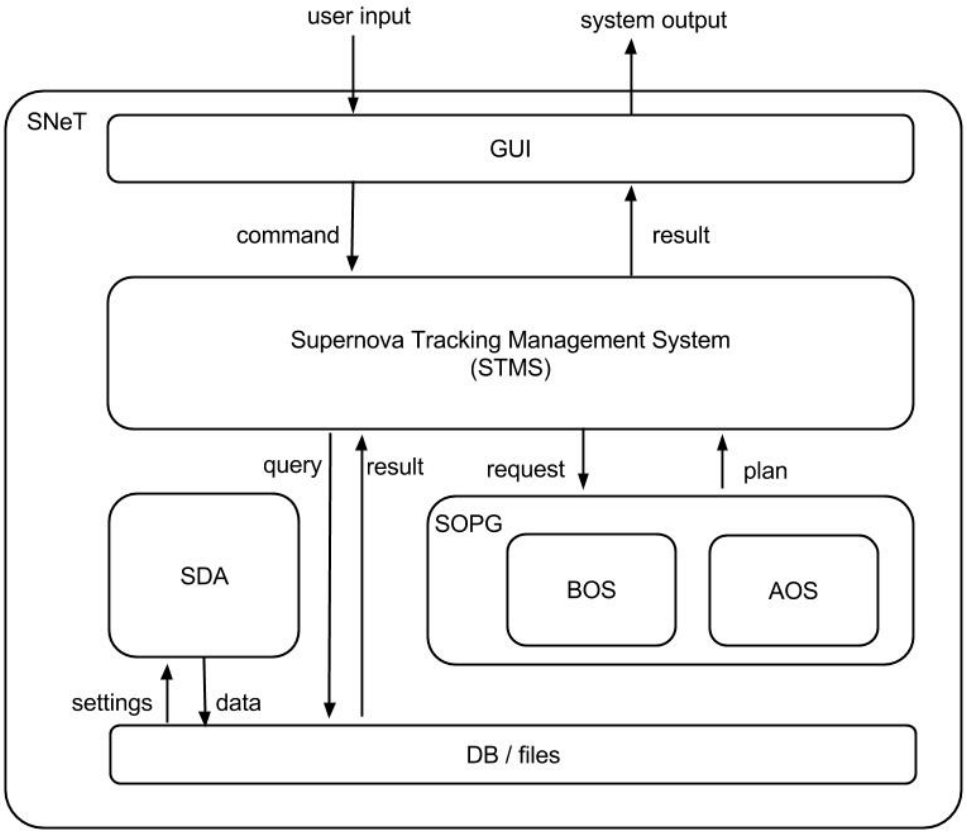


Figure 1.4.1 High-level view of SNeT

1.5 ROADMAP

In the next chapter we will discuss the SDA component. In Chapter 3 we will present the features of STMS. The design and implementation of BOS and AOS will be covered in detail in Chapter 4. In Chapter 4, we will also present an evaluation of our proposed scheduling algorithms. Finally, we will summarize our contributions and present our conclusions in Chapter 5.

2.0 SUPERNOVA DATA AGGREGATOR

The high level goal of the Supernova Data Aggregator (SDA) is to build a Supernova Data Repository to consolidate supernovae information from multiple sources (organizations) even though these sources may use different supernova naming convention, and provide different supplemental data (such as *redshift*, *magnitude*, etc.).

The design of SDA component is based on the idea of feeder aggregator. There are several applications sharing the same idea, like Google Reader [9]. These applications enable a user to access related information dispersed across multiple sites at one place. However, what makes our task of building SDA challenging is that (1) several sources distribute supernova related information in the form of messages or reports which are not well-formatted or well-organized and (2) different organizations use different formats such XML, semi-structured XML (XML combined with plain text) and HTML. Both of these facts point to the need of specialized parsing techniques that will enable the extraction and identification of "objects" from the different messages and reports and cross-match them. (Here and the following sections, the word "object" represents an entity of particular supernova with its associated parameters, such as *R.A.*, *Dec.*, *Redshift*, etc.)

2.1 SOURCE ANALYSIS

Currently, SDA collects documents in the form of messages from seven sources listed in Table 2.1.1. These documents are distinguished in terms of their format and content which in turn characterize the degree of difficulty in processing and integrating them.

The formats can be XML, ATOM, RDF/RSS, or HTML. In general, sources published by Skyalert are easiest to handle, as the information embedded in standard XML structure. Then comes the other Atom formatted source - CBET: Supernovae, because it can be considered as XML mixed with strings (the plain text part). The rest two sources are relatively hard to deal with, as HTML is the format for presentation instead of transmission.

Processing and integrating a document involves parsing the document, extracting the referenced "objects" in them and linking them to the "objects" in our repository. Documents may be limited to one "object" or reference multiple ones. In terms of the number "objects", the documents from the currently supported sources can be characterized as follows:

1) The Skyalert series - one entry represent one "object".
2) The Astronomer's Telegram: supernovae - one entry is an annotation on one or multiple "objects".
3) CBET: Supernovae - one entry represent one "object". Or, very rarely, it happens to be an annotation on a previous reported "object".
4) IAU Central Bureau for Astronomical Telegrams - there is no concept of "entry", but the HTML page provides a list of the "objects".

Sources	Format	Comments
Skyalert/CBAT: Central Bureau for Astronomical Telegrams	Atom - Atom syndication format (http://www.w3.org/2005/Atom)	Extended link with detail information, the linked page is in VOEvent v1.1 format (http: //www.ivoa. net/xml/VOEvent/v1.1)
Skyalert/CRTS: CRTS and SDSS Galaxy	As above	As above
Skyalert/CRTS2: Bright CRTS2	As above	As above
Skyalert/CRTS: CRTS and P60	As above	As above
The Astronomer's Telegram: supernovae	RDF - RDF Site Summary(RSS) 1.0 (http://web.resource.org/rss/1. 0/)	Extended link with detail information, the linked page is HTML
CBET: Supernovae	Atom - Atom syndication format (http://www.w3.org/2005/Atom)	Detail information in Plain Text
IAU Central Bureau for Astronomical Telegrams	Pure HTML page	N/A

Table 2.1.1 Sources supported by SDA

2.2 WORKFLOW OF SDA

The architecture of the aggregator is shown in the Figure 2.2.1. First of all, the “Feed Reader” operator maintains state information of each source, which includes the name of the source, its URL, the timestamp of the latest download along with the necessary credentials to access the source. It uses this information to download message/report entries from each source periodically.

The fetched messages/reports are stored in a database by the “Storing” operator. Further, the messages/reports, with detailed information acquired from extended link, if applicable, are passed to “XML Parser” operator and “Regular Expression or Regex Parser” operator, for the purpose of extracting the “objects”, collecting supplemental data associated with “objects” and creating an object record. Then any created record of “object” or “annotation on object” is inserted into the database, again by the “Storing” operator. Lastly, a crontab job (i.e., time-based job scheduler) is set at host node, running the whole routine periodically, guaranteeing the users to get newest supernova report in near real-time fashion.

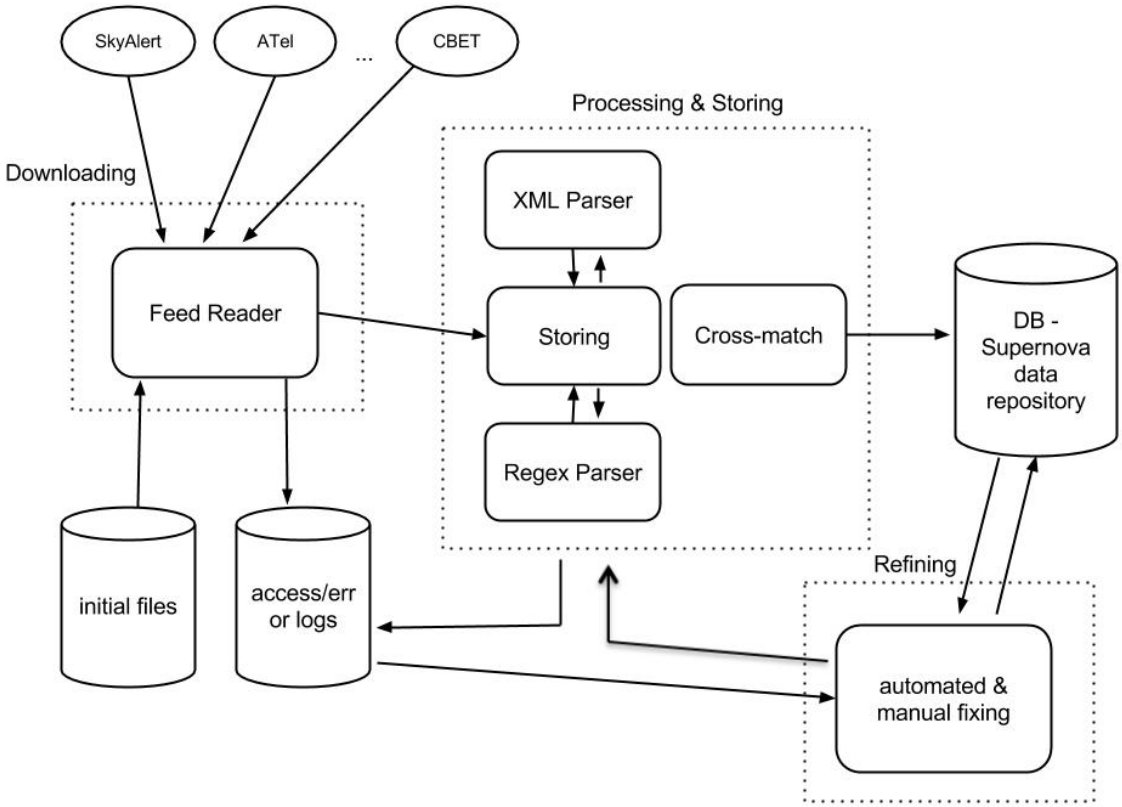


Figure 2.2.1 Illustrating the workflow of SDA

In addition, an access log and an error log are maintained for the aggregator. The access log records what source is accessed by SDA in exact what time, and how many updated entries of that source were downloaded. The error log records all the run-time errors such as the failure of downloading, parsing, or storing of message/report entries. In the case of parsing failure, a list of original messages/reports with the reason for the failure is saved and a notification is pushed to the administrator. The list of failed-to-parse messages/reports is also made available to the astronomer users, who can either submit a feedback on how to parse a specific message/report entry, or parse the specific message/report and manually store the extracted supernova data. In the former case the administrator uses the feedback to fix or enhance the parsing scripts. So, this "log, feedback, and fix" procedure makes up any potential drawbacks for "XML Parser" and "Regex Parser" to handle semi-structured data.

The operators in SDA are implemented in PHP (with the support of XML parsing library and regular expression library). The logs are stored in both files and a database (MySQL) tables.

2.3 BUILDING A SUPERNOVA DATA REPOSITORY

We use a relational database as the main storage of supernova data collected from different sources. The database schema is designed in a way that 1) every "object" can be linked/traced back to the original message/report entry and the source, and 2) "objects" retrieves from different sources can be cross-matched. Figure 2.3.1 shows the database schema of SDA in a UML format (table view only).

Most of the table names and their attributes are self-explanatory, except of Table *SN_objects* and Table *SN_uniques,* which need to be elaborated further. The former table,

SN_objects, stores original supernova "object" collected from sources, while the latter one, *SN_uniques,* maintains the "unique" supernova object in our repository, after the cross-matching of original "objects" based on their spatial position (namely the *R.A.* and *Dec.*). That is, if object A from source I and object B from source II are identical (i.e., referring to the same supernova), one "unique" object would be generated internally, to represent such supernova with multiple descriptions.

The "one to many" relationships from Table *SN_feeds*, to Table *SN_messages*, and then to Table *SN_objects* help in achieving the lineage goal, while the "many to many" relationship underlying in Table *SN_objects* and Table *SN_uniques* captures the cross-matching part.

The relational database used in implementing the supernova data repository is MySQL and SQL queries can be used to retrieve any supernova information in a consistent and efficient fashion.

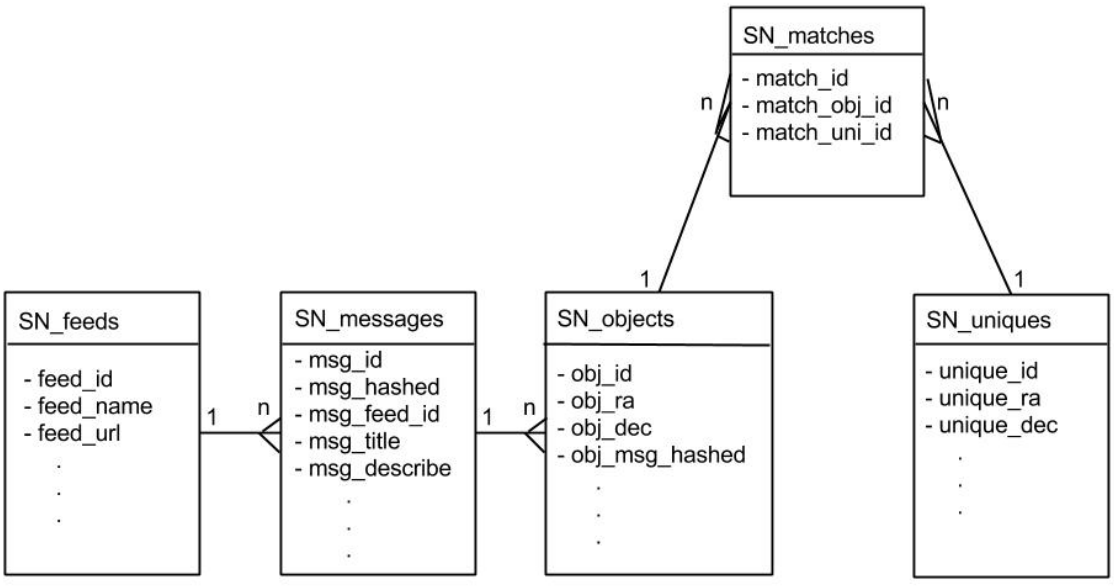


Figure 2.3.1 Database schema of SDA in table-view

3.0 SUPERNOVA TRACKING MANAGEMENT SYSTEM

As stated in the introduction, our Supernova Tracking Management System (STMS) helps users to organize their studies on supernovae, by providing them with interesting candidate supernovae, recording their ongoing observation plans, etc. With STMS, astronomers can efficiently manage their supernova study data online in an intuitive and easy way and concentrate on their research. STMS implements the interface of our entire solution, namely, the SNeT (computer-assisted SuperNovae Tracking) prototype system. STMS is also the "glue" in the whole system architecture, connecting with the aggregator SDA (discussed in the previous chapter, Chapter 2) and the scheduler components BOS and AOS (to be discussed in the next chapter, Chapter 4), interacting with the front-end GUI and the back-end database.

3.1 MAIN FEATURES OF STMS

Supernova Search/Browse The STMS can provide a list of interesting supernovae from our supernova knowledge-base for users. Users can list their search criterions via a GUI (Figure 3.1.1), then the STMS assembles the appropriate query for requesting the data from the database. Typically, users can search by supernova name, position range (*R.A.*, *Dec.*), or specify advanced filters on specific attributes such as *Mag.*, *redshift*, etc. Users are also able to control the order of

the search results and limit the number of results. The result is presented in a well-formatted and interactive table to the user (Figure 3.1.2).

List Management STMS helps users to build and manage their own lists of interesting supernovae, or simply called, lists of interest. By going through the search results generated from our system, users can select interesting supernova records and put them into their own working space, via a simple "drag & drop" operation. Then they can create and name a new list for those supernovae (Figure 3.1.2). Each user can have multiple lists of interest at the same time, and modify or delete them if needed (Figure 3.1.3). As a result, astronomer users can easily start from one of their lists and conduct follow-up experiments or observations.

Experiment Configuration STMS enables users to configure their experiments for planning observations. Typically, an experiment has three parts of input: 1) a list of interesting supernovae, 2) the constraints associated with each supernova in the list, and 3) a list of available nights (with how many hours) for the observation. We allow users to save/update the current experiment, or to bring a previous experiment into use (Figure 3.1.4). The night availability should be known ahead of time and the constraints on supernova stay constant. Thus, this feature reduces the repetitive work for users to configure an experiment, even for different observation lists. In addition, users can also make annotations on supernova's attributes, such as *type*, *magnitude*, *redshift*, *B-Peak*, etc. In the future, detailed, open-form annotations can be shared and exchanged among users, taking advantage of our Annotations Database [10, 11].

Plan Management Users manage and interact with their observation plans via the STMS. Once an experiment has been configured, the observation plan will be generated by the BOS and AOS components. The generated plan is presented in a table view with each row being a supernova in the list, and each column being a date (night), as configured by the user for this experiment (Figure 3.1.5). Several statistics like "Total Scheduled/Remaining Time", "Total Observed Time" are shown in the table, as well as observation "states" (i.e., whether to observe a particular supernova on a specific night or not) for all available nights. Meanwhile, extra statistics about whole plan are shown separately below the table (e.g., "Percentage of objects fully scheduled"). In addition, users can interact with the plan through AOS features. They can provide feedback to the current plan via "Like/Dislike" buttons, and switch to another plan by clicking "Try again". Or even more straightforwardly, they can edit the current plan in-place. When users check/uncheck cells in the table, the system will recalculate the schedulability and give a hint to the users (e.g., in case the total number of hours in a night exceeds the pre-allocated amount).

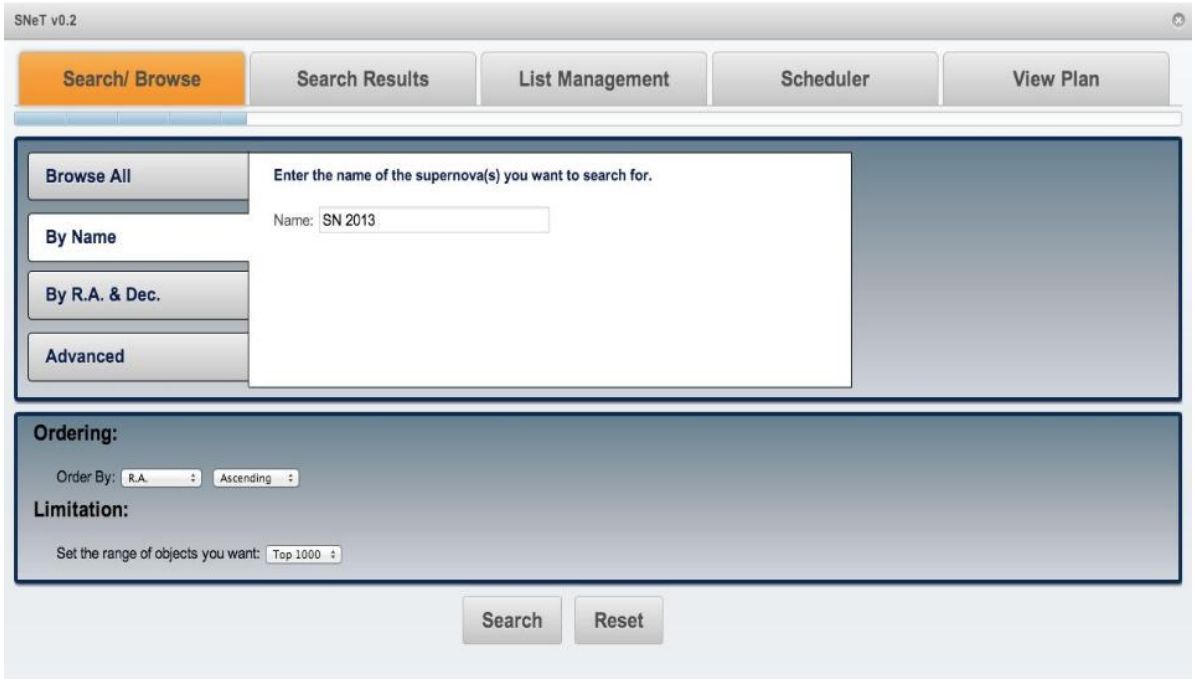


Figure 3.1.1 GUI of SNeT – Tab1

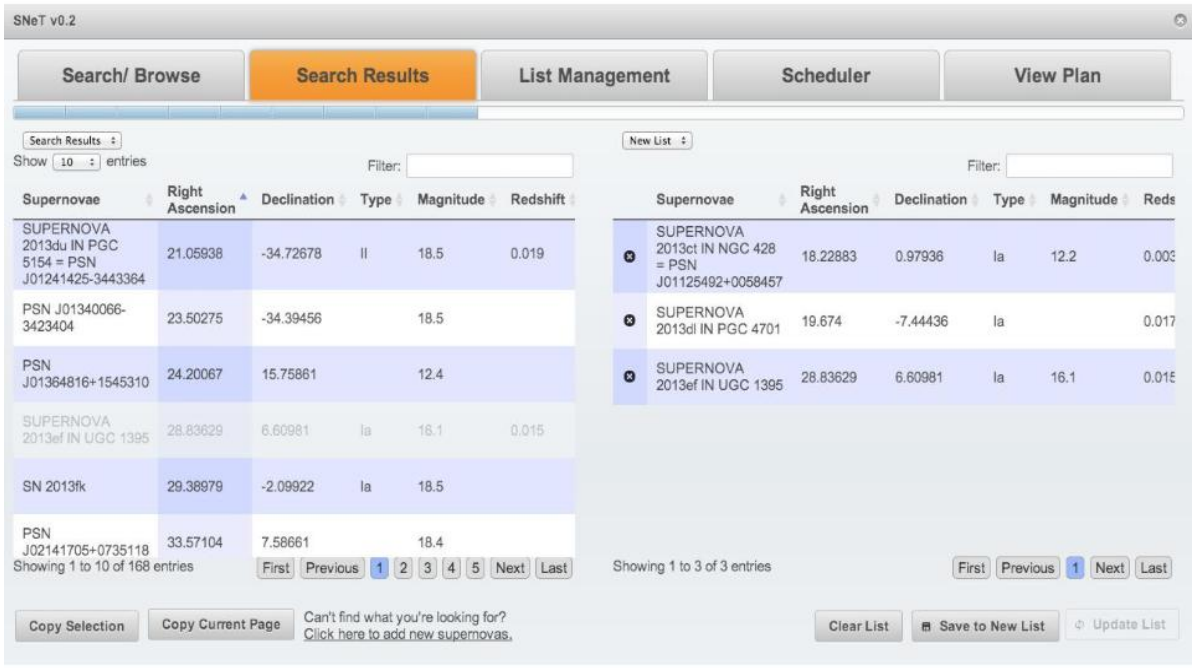


Figure 3.1.2 GUI of SNeT – Tab2

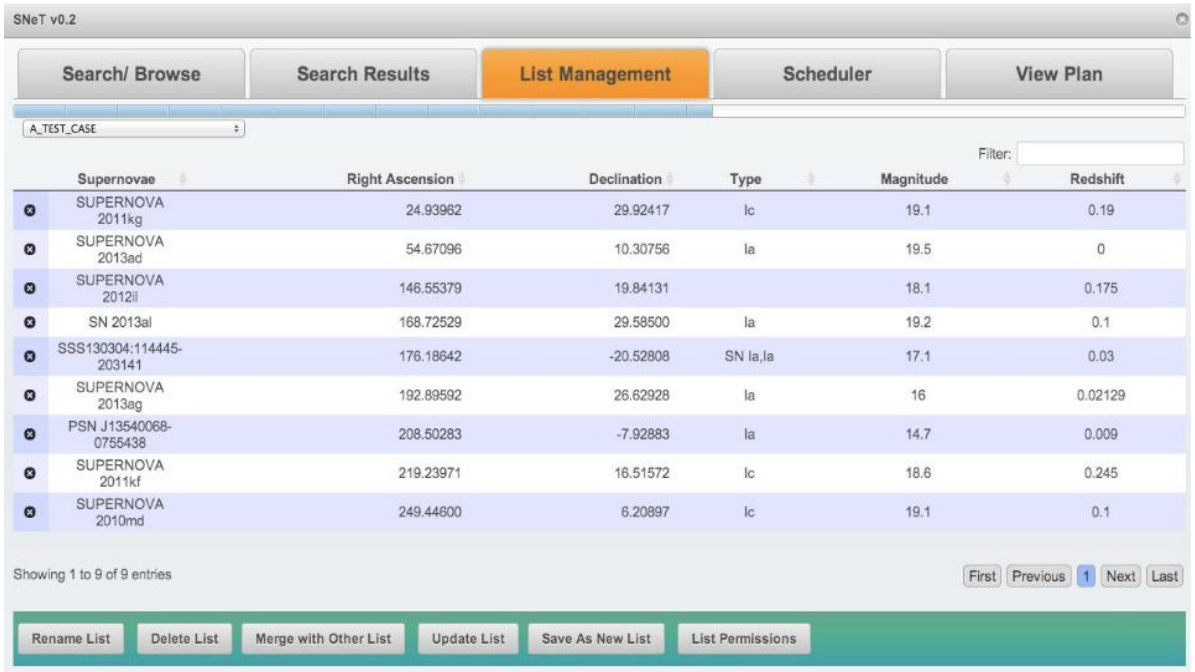


Figure 3.1.3 GUI of SNeT – Tab3

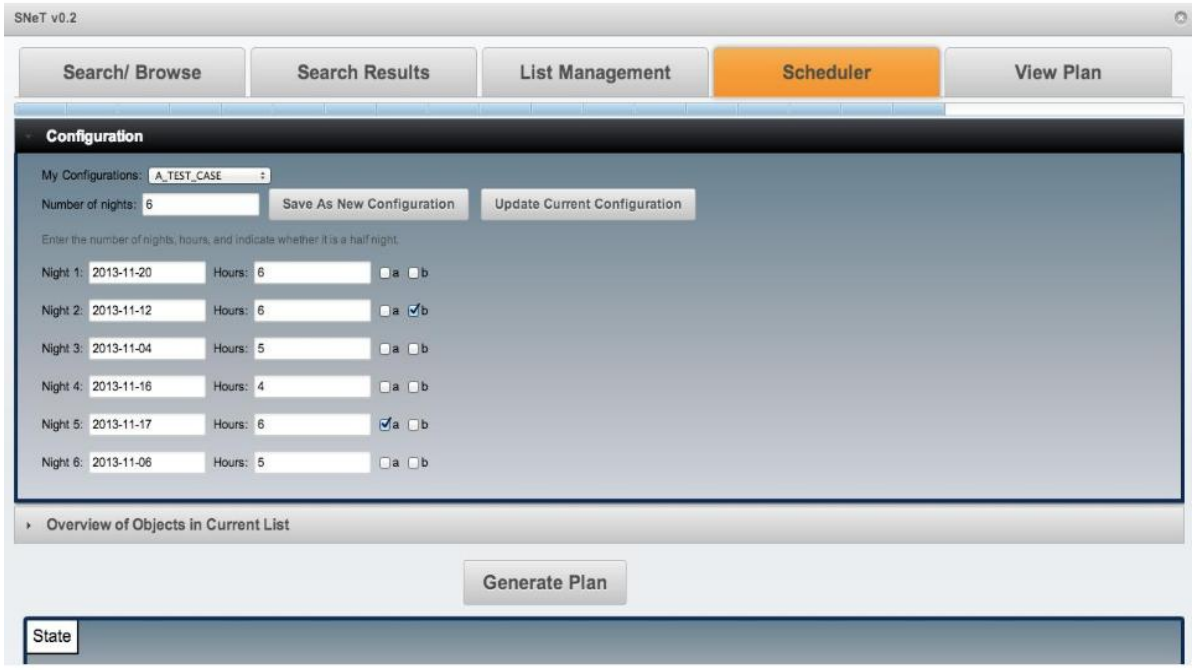


Figure 3.1.4 GUI of SNeT – Tab4

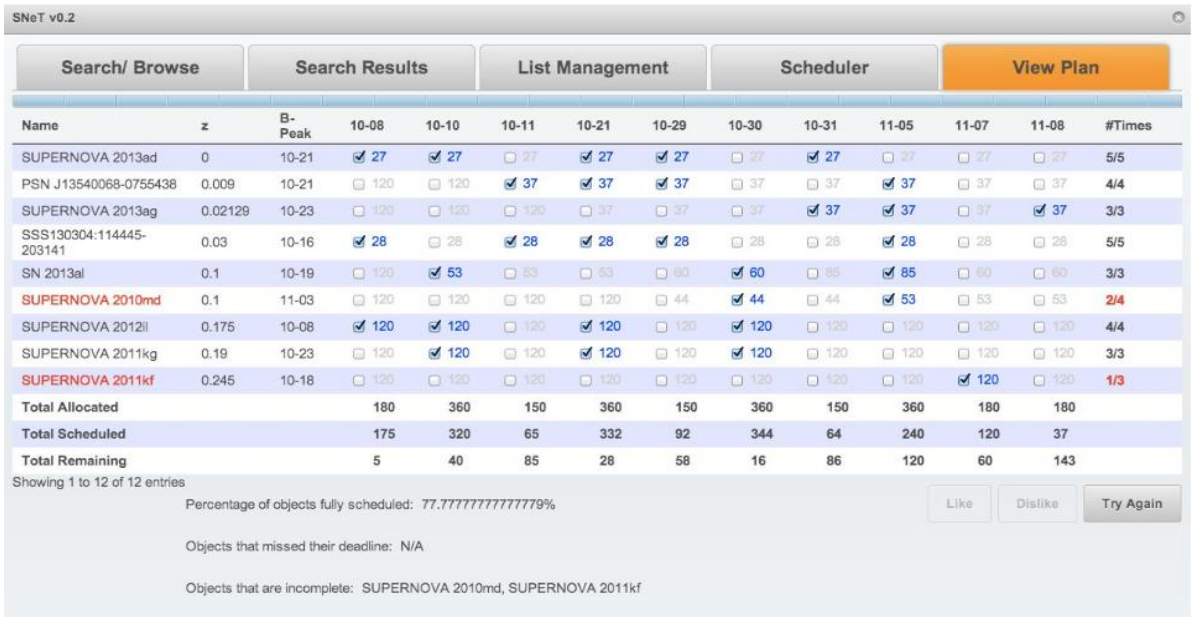


Figure 3.1.5 GUI of SNeT – Tab5

3.2 INTERACTING WITH THE GUI AND THE DATABASE

Essentially serving the role of "glue", the STMS is responsible for the communication between the front-end GUI and the back-end database. Users send their requests from the GUI, and the STMS transmits the request to the database and scheduler component. After that, the results are sent back to the GUI for users to see.

The GUI of SNeT has been integrated into the AstroShelf platform's web UI (Figure 3.2.1). A user has to log into the AstroShelf platform first for the services of STMS to be available. After a user log on, a "supernovae" button will appear in the upper-right tool bar of the web UI, and a dialog box will appear after he/she clicks the button. That dialog box is the GUI of SNeT, containing 5 tabs side by side, with the ordering of them indicating the natural process of assembling an observation plan. In other words, that 5 tabs are the interface to access STMS's 4 main features listed in the previous section.

Moreover, the database to support STMS is part of the whole database of SNeT. The relevant tables are shown in Figure 3.2.2. Table *SN_lists* and Table *SN_contains* are used to maintain lists of interest. Table *SN_exp*, Table *SN_exp_nights*, and Table *SN_exp_objects* are for the experiment configuration management. Table *SN_trains* is a training set, building up incrementally as users give feedback to the system, the records here will be retrieved for learning purposes to adjust the scheduling behavior. Finally, Table *SN_plans* is for the storing of positive observation plans, according to user's satisfaction. In addition, we should notice that: 1) there is a connection from the above tables to Table *SN_uniques* and beyond (other tables in the supernova data repository schema), binding the supernova data knowledge-base and working as a whole, and 2) there is a connection from the above tables to Table *Astro_user* and beyond,

indicating the integration with AstroShelf database and, more importantly, allowing STMS to provide user-specific services (e.g. a user-specific experiment).

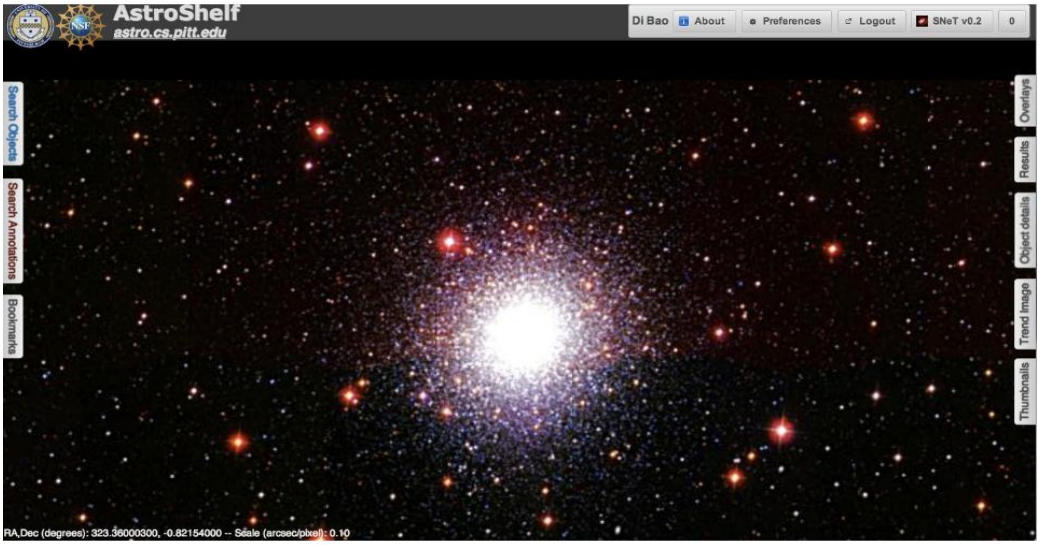


Figure 3.2.1 SNeT integrated into AstroShelf platform

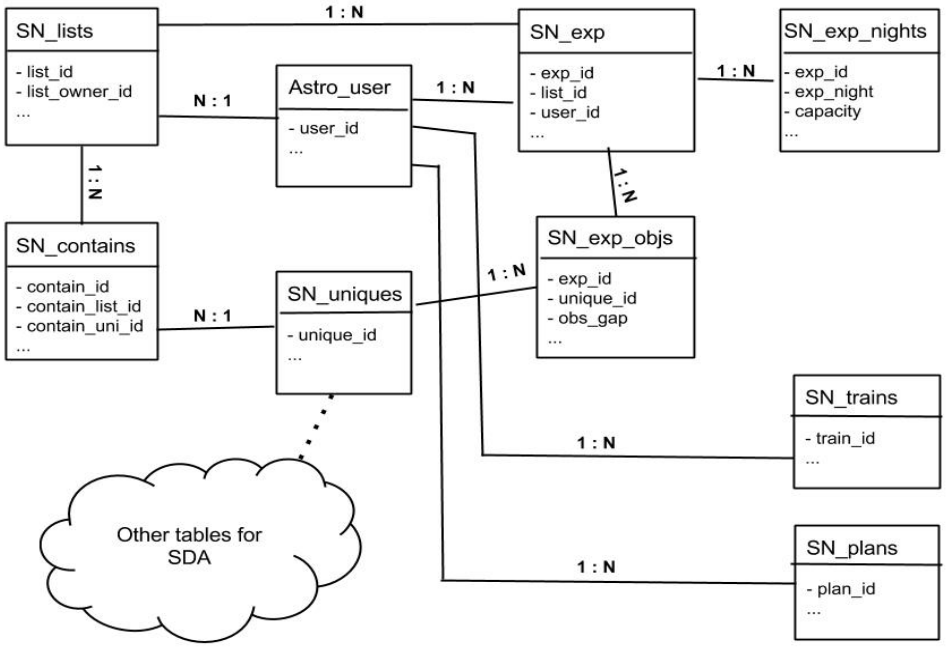


Figure 3.2.2 back-end database supporting STMS in table-view

4.0 SUPERNOVA OBSERVATION PLAN GENERATOR

In this chapter, we will cover the whole process of our automated supernova observation plan generator (SOPG) in detail. In Section 4.1, we will formalize the problem of "observation plan generation". In Section 4.2, we will introduce the high-level functional description about BOS and AOS. The workflow of "observation plan generation" by the collaboration of BOS and AOS will also be introduced here. After that, Sections 4.3 and 4.4 will dive into the essential parts of BOS and AOS, correspondingly, discussing the algorithms for planning. Lastly, Section 4.5 provides the experimental analysis about the core algorithms used by both BOS and AOS, and the overall performance of the planning system.

4.1 PROBLEM DEFINITION

We first define important terminology and then define the "observation plan generation" problem.

Def. 1 SN object *– A supernova object is characterized by its right ascension (R.A.), declination (Dec.), type, redshift, magnitude (Mag.), peak brightness (B-Peak), and priority. (Priority reflects a user preference on the supernova object, indicating how valuable the observation of such an object is. Thus, the observation of a low priority supernova object is more likely to be*

pruned out partially or completely to guarantee the overall schedulability.) An SN object SN_i is represented as a tuple:

$$SN_i = (RA, Dec, Type, Redshift, Mag, Phase, Priority)$$

Def. 2 Observation Constraints (OC) *– OC are associated with each particular SN object for a specific night (date). They include the start time (visibleS), the end time (visibleE), and the duration of exposure (obsDuration), measured in minutes, for its relevant supernova observation window. The OC are represented as a tuple:*

$$oc_i = (date, visibleS, visibleE, obsDuration)$$

Def. 3 Global Constraints (GC) *– Each SN object in the set has global constraints. An observation of a certain SN object is subject to the times it needs to be observed (obsTimes) and the gap between two consequential observation instances (obsGap), measured in days. The GC is denoted as:*

$$gc_i = (obsTimes, obsGap)$$

Def. 4 Night Map (NM) *– NM gives a list of nights available for observing supernovae and the exact hours available for the observations. It is denoted as an associative array:*

$$NM = \{night_1 : hours_1, \dots, night_k : hours_k\}$$

Def. 5 Plan (P) *– Plan P is the result of the scheduling. It specifies what supernovae to observe and how long the observation instances for the supernovae should be, for each night in the NM. It is represented as follows:*

$$P = \{night_1 : list_1, \dots, night_k : list_k\}$$

$$list_i = \{(SN_i, OC_i), \dots, (SN_j, OC_j)\}$$

Given the above definitions, the inputs of the problem are:

- A target list of supernova objects, $S = \{SN_1, \dots, SN_n\}$,
- A list of observation constraints, $OC = \{oc_1, \dots, oc_n\}$,
- A list of global constraints, $GC = \{gc_1, \dots, gc_n\}$, and
- A night map, NM.

The output of the problem is a plan P. Thus, if we present the whole "observation plan generation" problem as a function of the *OP* (Observation Plan), our task is formalized as:

$$OP(S, OC, GC, NM) = P$$

The description of the problem of "observation plan generation" is as follows. Given:

- A target list of supernovae, each of which contains full information about its *right ascention*, *declination*, *type*, *redshift*, *magnitude*, *peak brightness date* (or the exact phase plus exact discovery date);
- A list of observation constraints, each one per each supernova in the list, indicating the observation window for a specific night (date);
- A list of global constraints, each one applied on one particular supernova to be included in the observation plan, requiring particular times of observation and gaps between each two consecutive observation instances;
- A list of nights available for conducting observations and the specific hours available in each listed night,

Find a feasible and reasonable observation plan specifying what supernovae to observe for how long for each listed night, that satisfies all the above conditions if possible, or at least, maximizing data acquisition, ensuring the quality of observation data gained, with respect to user preferences on the supernovae in the given list.

4.2 HIGH-LEVEL VIEW OF PLANNER

A planner consists of two core components named BOS and AOS, and 4 layers, which are the local scheduler in BOS, the global scheduler in BOS, the "space" reserving layer in AOS, and the semi-supervised learning layer in AOS. The structure of the planner is illustrated in Figure 4.2.1. The local scheduler is responsible for the scheduling of available observation instances in a particular night. The global scheduler is responsible for the scheduling of observation tasks in the whole timeline. The "space" reserving layer generates appropriate night capacity distribution to reserve a certain amount of capacity, in anticipation of future updates to the list of interesting supernovae and in cases of unsuccessful observations (e.g., due to cloudy skies). The "learning" layer generates weights using machine learning to tune the influence of factors in local/global scheduling. The following sections will give more detailed descriptions of the functions, features, and principles of each layer. In this section, we only explain, in a top-down way, the design purpose and workflow of the SOPG.

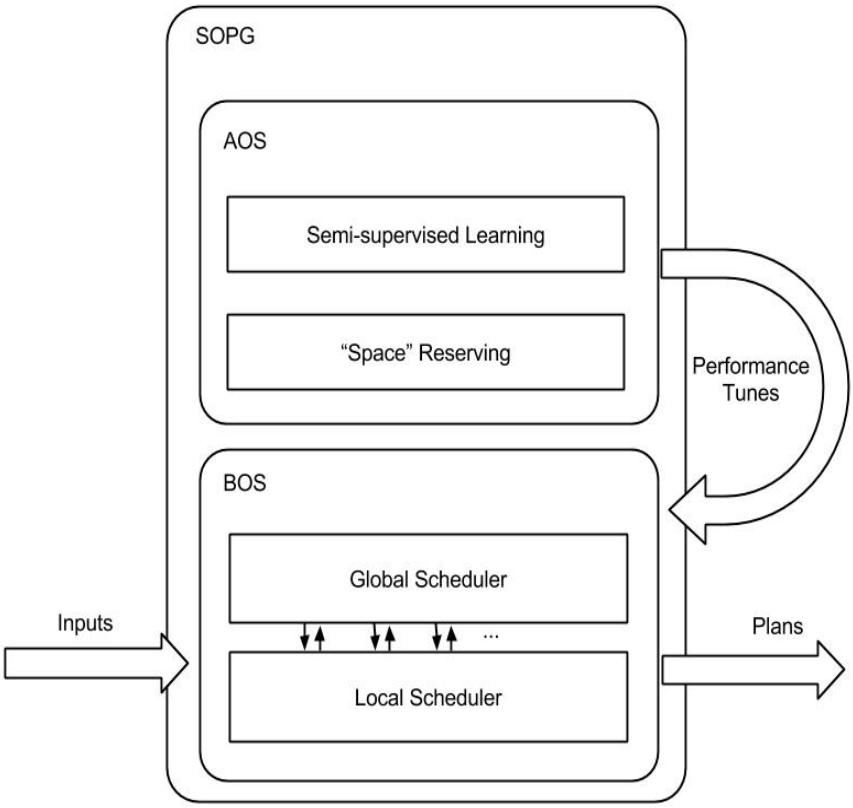


Figure 4.2.1 High-level structure of SOPG

Let us first introduce the two different "views" (abstractions) of the planning process. One way to view the process is that it considers a set of bins with certain capacity (Figure 4.2.2), and then it assigns items with different values and weights into the bins in a way to gain the maximum profit. In our supernovae tracking scenario, each night can be considered as a "bin", the available hours in each night are our "capacity", and of course the observation task (instance) turns into "item with different value and weight". In addition, the temporal constraints for one instance or between instances make the planning selection even more complicated to perform with.

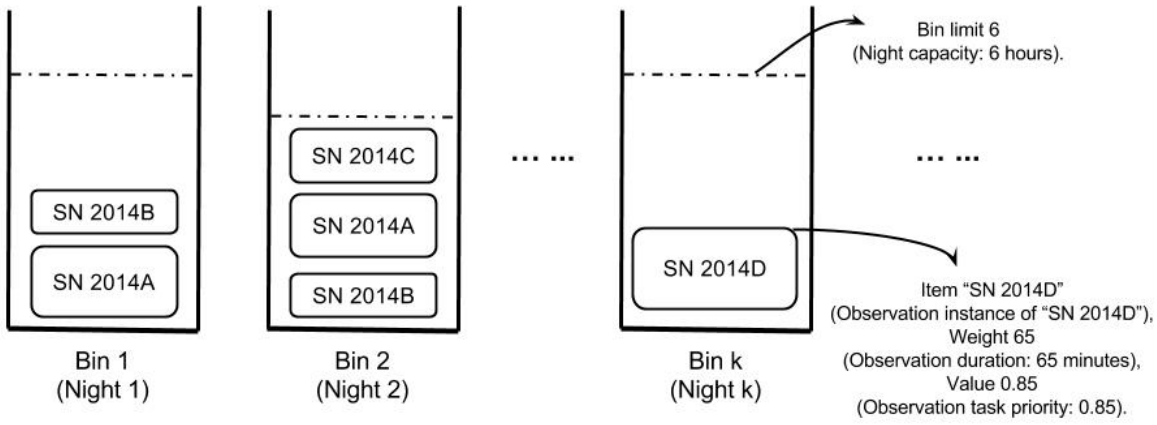


Figure 4.2.2 View 1 – Bin packing

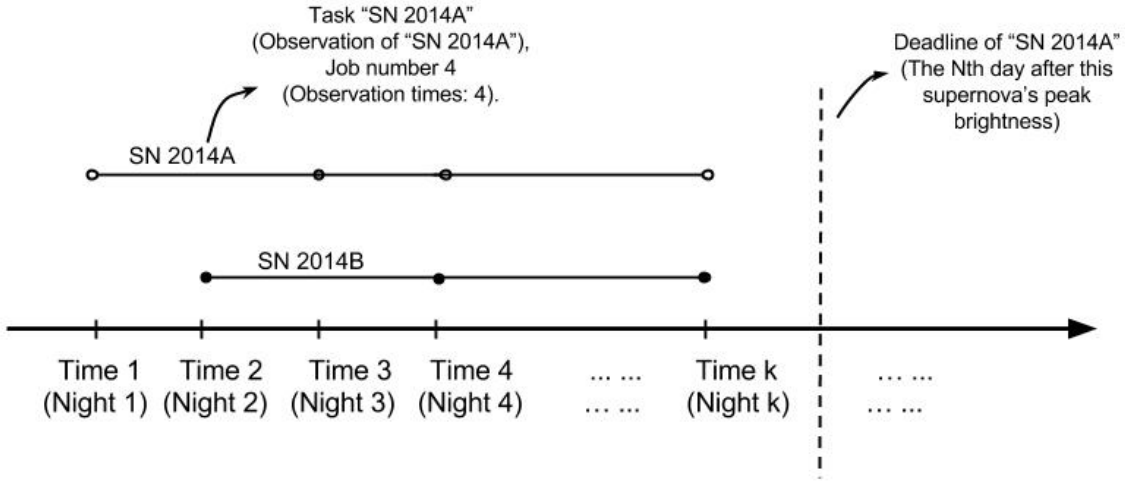


Figure 4.2.3 View 2 – Scheduling with soft deadline

The other way to view the planning process is as scheduling a set of tasks with soft deadline given a certain time interval (Figure 4.2.3), schedule the execution of a set of tasks with

soft deadlines. Obviously, the observations will be tasks to be scheduled, the number of instances will be quantified as the workload, and the brightness peak of each supernova object can be used as soft deadline.

As the problem can be viewed in two ways, a failure in either scenario will degrade the quality of plan. For example, "failure to put all items into bins" indicates that certain observation task remains incomplete, while "failure to execute and finish all tasks before their deadlines" indicates that some observation tasks gained low quality data, since they got to be observed outside their optimal time window.

These two abstractions underlined the design principles of our planning system. Specifically, the local scheduler in BOS and the "space" reserving layer in AOS are implemented for solving the problem based on the first view of bin packing [12]. On the other hand, the global scheduler in BOS is implemented based on the second view of scheduling soft deadline tasks [13]. In addition, the learning layer in AOS added the semi-supervised learning feature into the system to decrease the failure rate in both views.

So, the system's planning process in a top-down view is:

1) *The learning layer generates the weights on factors based on the training set,*
2) *The "space" reserving layer calculates a capacity distribution across the pre-specific set of nights (dates),*
3) *The global scheduler sorts the available observation tasks in each night, taking the most urgent portion as the candidates for that night,*
4) *The local scheduler picks the most valuable portion of the candidate set from step (3) and schedules them for the particular night,*
5) *Repeat Steps (3) and (4) for every night iteratively to generate a complete plan.*

Notice that we usually refer to Step (3) as “global” since it views the process in a night-based way, while referring to Step (4) as “local” since it views the process in an hour-based way. The relationship of the 4 layers behind the scene is that the top two layers in AOS provide necessary configuration data. The bottom two layers in BOS generate the plan collaboratively. One more thing needs to be pointed out is that BOS can work properly without AOS (using default system configuration instead of whatever necessary input data from AOS).

In addition, the actual deployed system interacts with a client-side user interface to generate the plan iteratively. The user can stick to the current input requirement to go through all the possible plans and comment on them one by one. The planner will record the feedback from the user for building and maintaining a training set, which is crucial for the learning layer.

The pseudo code of the planner is described in Algorithm 4.2.1. In the algorithm, the term “L0Scheduler” represents the local scheduler in BOS. The term “L1Scheduler” represents the global scheduler in BOS. The term “L0Iterator” stands for the “space” reserving and night capacity distributing layer. The term “L1Iterator” stands for the semi-supervised learning layer. We consider the interaction with a client-side user as another “layer” for the planning system (termed as “part I” in the algorithm).

Finally, a detailed implementation-level description of “SN Class” has been given in **APPENDIX A**. As the basic entity enrolled in the process of plan generation, “SN Class” provides an insight look of the system.

Algorithm 4.2.1

Part I (very outer layer):

Take user input from client-side, configure and initialize the planner.

While loop: // loop until user finalizes a plan

Call main_procedure(), the execution procedure of 4 layers.

If returned plan **P** is empty:

Break the loop and hint the user.

End if

Organize generated plan **P** and send back to client-side.

Prompt to get user's feedback **F** of current plan.

Store **F** with current weights **W** into training set.

Prompt to get user's instruction to **continue** or **quit**.

End while loop

Part II (4 inner layers):

Func main_procedure():

Call L1Iterator to generate weights **W** by learning from training set.

Call L0Iterator to get capacity distribution for night map **M**.

If capacity consumption overflow:

Fail main_procedure() and return empty plan **P**.

End if

For each night **N** in **M**:

Call L1Scheduler to get candidate set **C** of obs_instances.

Call L0Scheduler to get final set **F** obs_instances.

Store the pair **N=>F** for this night's plan.

End for loop

Return plan **P** as a set of pairs **N=>F.**

4.3 BASIC OBSERVATION SCHEDULING

The Basic Observation Scheduling component (BOS) consists of two phases, with different scheduling granularity. Phase one is "local scheduling", which schedules at the granularity of hours in a particular night. Phase two is "global scheduling", which schedules at the granularity of days, over the set of all pre-specified dates. We will discuss the local scheduling first and then discuss the global scheduling, in the following subsections.

4.3.1 Local Scheduler

The Local Scheduler is the foundation of the whole SOPG system, as it makes the final decision on which supernovae to observe in which night. The problem it solves can be described as following:

Given a candidate list of supernovae, select the most valuable (according to the priority provided by user) portion from the list, while satisfying two safety conditions:

C1 *The sum of the durations of the selected observation instances does not go over that night's capacity limit.*

C2 *Every observation instance selected can be finished within its corresponding available window (in that night) without any overlapping or conflicts with others.*

In other words, the problem is a "combinatorial optimization" one [14, 15] and the goal is to find a feasible schedule to maximize the overall "profit". However, achieving local optimality in every step does not always lead to global optimality. Therefore, we define heuristic functions over multiple factors and determine weights for these factors dynamically to handle this situation.

4.3.1.1 Reduction to 0/1 Knapsack

Intuitively, our problem is related to the Knapsack problem [16] (if we do not consider the temporal constraints as condition **C2** states):

Given a set of items, each with a mass and a value, determine the number of each item to include in a collection, so that the total weight is less than or equal to a given limit and the total value is as large as possible.

In our supernovae tracking scenario, an "item" is an observation instance, the "mass" is the duration of that instance, the "value" is its priority, and the "limit" is the capacity of night. As we cannot pick the same item multiple times, the counterpart is actually 0/1 Knapsack. This kind of problem is called a weakly NP-complete problem [17, 18], which indicates that it is hard to solve, but is still solvable within a reasonable amount of time. So one main focus of our local scheduler is to develop feasible approaches applicable to our problem.

As the Knapsack problem can be solved using approximation algorithms [19] and dynamic programming [20], we developed our approaches referring to those techniques. It should be pointed out that our problem is more complicated than the 0/1 Knapsack problem due to the extra temporal constraints. There might be cases in which the sum of duration of two instances fit into the remaining capacity but their observation windows are overlapped, which can shrink the actual usable capacity. We discuss how to validate a given schedule by checking **C2** in subsection 4.3.1.4.

4.3.1.2 Local Heuristic

The main reason why we need to develop a local heuristic is that our problem is more complicated than the Knapsack problem. A naïve greedy approach with a criterion like

$goodness = \frac{value}{weight}$ is problematic because it does not take the temporal constraints into account. So, besides the evaluation of goodness, we also need to evaluate the "fitness" of the "item", to rule out the fewest possible choices of the candidate list that would violate the constraints and still leave the maximum flexibility. An "item" with a higher fitness score is considered less likely to have a conflict with the others. There are two factors having influence on "fitness":

1) The observation window length,

2) The accumulated overlapping length of the current window and the other windows.

Clearly, a big observation window increases the flexibility for the instance to fit in, especially when its duration is much smaller than the window length. On the other hand, a small overlapping length decreases the potential risk of conflict. Given the above, the local heuristic is devised as follows (Equation 4.3.1):

$$\boldsymbol{H_{local}(SN) = \frac{100*prority+w_1}{duration+w_2} + \frac{win\_len + w_3}{overlap\_len + w_4 + 1}} \quad \textbf{(Eq. 4.3.1)}$$

In the equation, $w_1 \sim w_4$ are the weights associated with the corresponding factors. These weights are used to amplify or reduce the effect of one of the factors to scheduling decisions. The weights are designed to capture: 1) the uncertainty from local optimality to global optimality, 2) the ambiguity in a user's measurement of goodness. Point 1 has been discussed previously, and for point 2, we should allow the diversity of users' preferences on plans (e.g., for the same plan, user A thinks it is good while user B thinks it is bad). So the weights act as "side effect" or "noise" to tune the behavior of the scheduler to handle the stated uncertainty and ambiguity. As a whole, the heuristic provides the system a consistent way to measure the goodness/fitness of a target. In the local scheduling algorithms introduced below, such a heuristic plays a crucial role in decision-making.

4.3.1.3 Local Algorithms

1) Baseline Algorithm

The most trivial approach is to enumerate in a brute-force way all the possible combinations of observation instances, then eliminate those that violate conditions **C1** and **C2** (mentioned at the beginning of section 4.3.1), and from the remaining ones consider those with the local maximum value, and repeat until global optimality is reached. Assuming there are N objects considered, the runtime complexity of this algorithm is (Equation 4.3.2):

$$\boldsymbol{complexity} = \boldsymbol{O}(\boldsymbol{2^N}) * \big(\boldsymbol{O}(\boldsymbol{N}) + \boldsymbol{O}(\boldsymbol{N})\big) \approx \boldsymbol{O}(\boldsymbol{N} * \boldsymbol{2^N}) \quad \textbf{(Eq. 4.3.2)}$$

There are 2^N combinations of candidate objects. For each combination, we need to validate it against conditions **C1** and **C2**, which would cost at least $O(N)$ time. So the baseline algorithm will become exponentially costly as the candidate list grows. Although cost-prohibitive for the general case, this trivial approach makes for a very good baseline for our comparison experiments later, as it does not require approximation or heuristics. In addition, it is still a feasible and optimal algorithm when the problem set is small, which indicates that we can mix it with our proposed algorithms to handle base-cases. This baseline algorithm is presented in Algorithm 4.3.1.

2) Greedy Beam Search

This algorithm is very similar to the naïve greedy approximation. Using the heuristic proposed in subsection 4.3.1.2 as a greedy criterion, the algorithm runs a beam search [21] in the state-space (structured as a tree) of possible alternative choices, only keeping a fix-length fringe (i.e., fixed number of successor states) for branching. Assuming the height of the tree is H, the average

branching factor is b, and the beam width is w, the runtime complexity is calculated in Equation 4.3.3:

$$\boldsymbol{complexity = O(H) * O(wb) \approx O(wb * log_b 2^N) \approx O(wb * N)} \quad \textbf{(Eq. 4.3.3)}$$

Intuitively speaking, this algorithm keeps multiple promising local sub-solutions to increase the chance of reaching a global optimal solution. This algorithm actually turns out to have good performance in practice. The details of this algorithm are shown in Algorithm 4.3.2.

3) Iterative Dynamic Programming

The 0/1 Knapsack problem is weakly NP-complete and can be solved using dynamic programming [22]. The main sequence of steps is as following:

1. Build a two-dimensional array V[index, weight]. (The "index" represents the first index amount of items to be considered; the "weight" is the current weight limit; and the value of each array entry is the maximum value to be gained. E.g., V[i, w] = k is interpreted as: "for the 0 to ith elements, the maximum value that can be achieved is k, with the total weight not going over w".)

2. Initialize the array, V[0, 0] = 0, V[0, w] = 0, V[i, 0] = 0.

3. Iterate through the two-dimensional array, filling each entry with index > 0 and weight > 0, according to the rule: V[i, w] = MAX(V[i – 1, w], value of the ith item + V[i – 1, w – weight of the ith item]).

4. The bottom-right entry's value will be the maximum value to be gained for the original problem.

This is a standard dynamic programming algorithm that we are familiar with. However, in such a process, condition **C2** is not checked at all, and thus the returned answer may not be feasible due to the time window conflict. Therefore we take an iterative DP approach which runs DP multiple times. When an intermediate answer violates condition **C2**, the algorithm will select a victim to kick out and restart. The victim is chosen by evaluating the local heuristic. Theoretically, we can run as many iterations as possible, but we cut off the process by a threshold to trade some accuracy for the overall performance. Assuming there are I runs in total, N items in the set, and weight limit W, the runtime complexity of this algorithm is shown in Equation 4.3.4:

$$\boldsymbol{complexity} = \boldsymbol{O}(\boldsymbol{I}) * \big(\boldsymbol{O}(\boldsymbol{NW}) + \boldsymbol{O}(\boldsymbol{N}) + \boldsymbol{O}(\boldsymbol{N})\big) + \boldsymbol{O}(\boldsymbol{N}) + \boldsymbol{O}(\boldsymbol{N}) \approx \boldsymbol{O}(\boldsymbol{IWN}) \quad \textbf{(Eq. 4.3.4)}$$

The standalone $O(N)$ time is for either condition **C2** validation or for choosing a victim. The algorithm is more stable than the previous (and the next one as well), as it ensures the optimality of the answer once the constraint validation is passed. The details of this algorithm are shown in Algorithm 4.3.3.

4) Random Restart Hill Climbing

As we know, in computer science, local search [23] is a method for solving optimization problems that are computationally expensive. Local search can be used on problems that can be formulated as finding a solution that maximizes a criterion among a number of candidate solutions. Local search algorithms move from one solution to another solution in the space of candidate solutions.

Our problem can be viewed as an optimization problem - selecting the best object (or portion) from a group of candidates. For a particular night, if we arbitrarily select a subset (portion) of objects from the candidate list as the initial state, then the problem becomes how to

walk from the current state to the goal state (with the best objective function value). By saying "walk" or "wander", we mean swapping a less valuable object out for a more valuable one in. However, as a basic local search algorithm, hill-climbing has the drawback of getting trapped into local maximums, plateaus, or ridges, if it happens to select a bad subset of objects as starting point. So we improve our algorithm by randomly restarting the hill climbing process for several number of times: i) start different hill-climbing searches from random starting positions and stop when a goal is found, ii) save the best result from the explored states. If all states have equal probability of being generated, a goal state will eventually be generated with probability approaching 1 by selecting random initial state and repeat this algorithm.

The specific algorithm used in our scenario can be described in the following steps:

1. Set sufficient restart times, and repeat Steps 2, 3, 4 accordingly.

2. Randomly pick a subset of objects from the candidate list, as the initial state.

3. Go through the rest of the candidate list. Add additional objects in if they do not violate the constraints.

4. For each pair of objects in the current state and the rest of the candidate list, make a move (swap) if necessary, until a goal is found.

5. For all the goal states found by executing Steps 2, 3, 4, save the best result.

Assuming that the restart times is K, the candidate list size is N, the runtime complexity of this algorithm is calculated in Equation 4.3.5:

$$\boldsymbol{complexity} = \boldsymbol{O}(\boldsymbol{K}) * \left(\boldsymbol{O}(\boldsymbol{N}) + \boldsymbol{O}(\boldsymbol{N^2}) + \boldsymbol{O}(\boldsymbol{N^3})\right) \approx \boldsymbol{O}(\boldsymbol{K} * \boldsymbol{N^3}) \quad \textbf{(Eq. 4.3.5)}$$

Compared with the previous two algorithms, this algorithm seems not ideal since it runs in polynomial time with a big coefficient (could be as big as N). But in practice, this algorithm does not lose any performance or accuracy compared with the previously described algorithms. The details of this algorithm are presented in Algorithm 4.3.4.

Algorithm 4.3.1

```
Func brute-force():
    OPT = None.
    For each combination (subset) from candidate list:
        If violate constraint 1 – weight overload:
            Continue.
        End if
        If violate constraint 2 – window conflict:
            Continue.
        End if
        If current value > OPT:
            OPT = current subset.
    End for loop
    Return OPT
```

Algorithm 4.3.2

Func greedy-beam-search():

Calculate and decide beam width **B**.

Set initial state to be empty.

Set an empty set **SG** for "goal states".

Add initial state on the fringe **F**.

While sizeof(**F**) > 0: // keep branching if fringe is not empty

Set new fringe **F'** as an empty fringe.

For state in current fringe **F**:

Generate all the successor states into set **S**.

For each state in **S**:

Do forward checking:

1) Weight overload 2) Window conflict

If forward checking fails:

Delete the state and **Continue**

Else:

Add the successor state into **F'**.

End if

End for loop

If set **S** is empty:

Add current state into **SG**. // A state cannot add more element

End if

If lengthof(**SG**) == **B**: // if enough goal states been found

Break while loop.

End if

End for loop

Sort new fringe **F'** using heuristic $\boldsymbol{H_{local}}$.

Take the first **B** elements in **F'**.

F = F'

End while loop

Return the most valuable one in **SG**.

Algorithm 4.3.3

Func iterative-DP():

Calculate and decide iteration times **I**.

Set the initial set **S** with all the candidate objects.

While I > 0:

Set up two-dimensional arrays **V**[index][weight] and **K**[index][weight]. **K** is used for backtracking.

For i from 0 to max_index: // the max_index varies, depending on size of **S**

Get the **i**th item's priority as **value**.

Get the **i**th item's duration as **weight**.

For w from 0 to max_weight:

Update entry **V[i][w]** = MAX(**V[i - 1][w]**, **value** + **V[i - 1][w - weight]**).

Update entry **K[i][w]** to 1 if pick the current item.

End for loop

End for loop

Backtrack to get the current answer **C**.

If C not violate constraint 2):

Return C

Else:

Choose victim with minimum $\boldsymbol{H_{local}}$ and delete from **S**.

End if

I = I -1

End while loop

// if run out of iteration times, we will directly delete item from current answer

While not satisfying constraint 2):

Kick out victim with minimum $\boldsymbol{H_{local}}$ from **C**.

End while loop

Return C

Algorithm 4.3.4

Func random-restart-hill-climbing():

Calculate and decide restart time **K**.

Set current state **S** to be empty.

Set the rest set **R** to have all items in candidate list.

Set global state **GS** to be empty.

While restart time **K** > 0:

While not violating constraint 1) weight overload 2) time window conflict:

Random select object in **R,** add into **S** and delete from **R**.

End while loop

If isEmpty(**R**) is **True**:

Return S. // special case, the whole candidate list can be scheduled.

End if

For each object in **R** in the order **sorted** by $\boldsymbol{H_{local}}$:

If not violating constraint 1) and 2):

Add the object into **S** and remove from **R**.

End if

End for loop

While existing better successor state:

For each object $\boldsymbol{O_1}$ in **S**:

For each object $\boldsymbol{O_2}$ in **R**:

If value of $\boldsymbol{O_1}$ < value of $\boldsymbol{O_2}$ and not violate constraints:

Swap $\boldsymbol{O_1}$ out for $\boldsymbol{O_2}$.

End if

End while loop

Replace **SG** with current **S** if **S** has higher value.

Reset current state **S** and the rest set **R**.

K = **K** - 1

End while loop

Return GS.

4.3.1.4 Greedy Validation

Up till now, there is one very important issue that has not been thoroughly examined yet. Validating condition **C1** could be easy, but how to validate condition **C2** (the time window conflict) efficiently? Let us review the problem first: there is a list of observation instances, each needing certain amount of time to finish. Meanwhile, each instance must be executed within a specific time window (which depends on the supernova's *R.A.*, *Dec.*, and the location of the observations). Then how can we know whether all the instances can be finished without any time conflict?

Actually, the problem description above is a good fit for classic scheduling problems. The formal definition of this problem is:

Assume a set of tasks $\{T_1, T_2, \dots, T_n\}$*, each of which has a release time* t_r*, a deadline* t_d*, and a worst-case execution time* E*. Each task being scheduled to start at time* t_s *should satisfy:*

1) non-preemptive execution,

2) $t_s > t_r$*, and*

3) $t_s + E < t_d$.

How can we schedule all the tasks, or can we confirm the schedulability of those tasks?

Note that task execution cannot be interrupted. Otherwise, an EDF [24] scheduling algorithm will be optimal (i.e., can guarantee that no task will miss its deadline if the task set is schedulable), and we can design an algorithm to determine the schedulability in linear time for the worst case. However, EDF is not suitable in non-preemptive scheduling. Furthermore, we cannot determine which task to execute first according to neither start time nor deadline. Figure 4.3.1 (a) and (b) show, separately, the failures caused by scheduling in either "first start first

server" or "earliest deadline first" way, even though optimal schedules exist for both case. Actually, for N tasks, there are $N!$ ways to schedule their execution, all of which are likely to be feasible. This makes determining the schedulability of those tasks an NP-hard problem. On the other hand, special cases such as (a) and (b) in Figure 4.3.1 are rare, as the time window has some spatial locality property based on a supernova's position. As such, it is not worth taking a brute-force approach for schedulability validation. Without any better criterion for greedy approximation, we schedule in the first-come-first-serve manner based on the windows' start times. In this way, the validation of condition **C2** is checked in linear time.

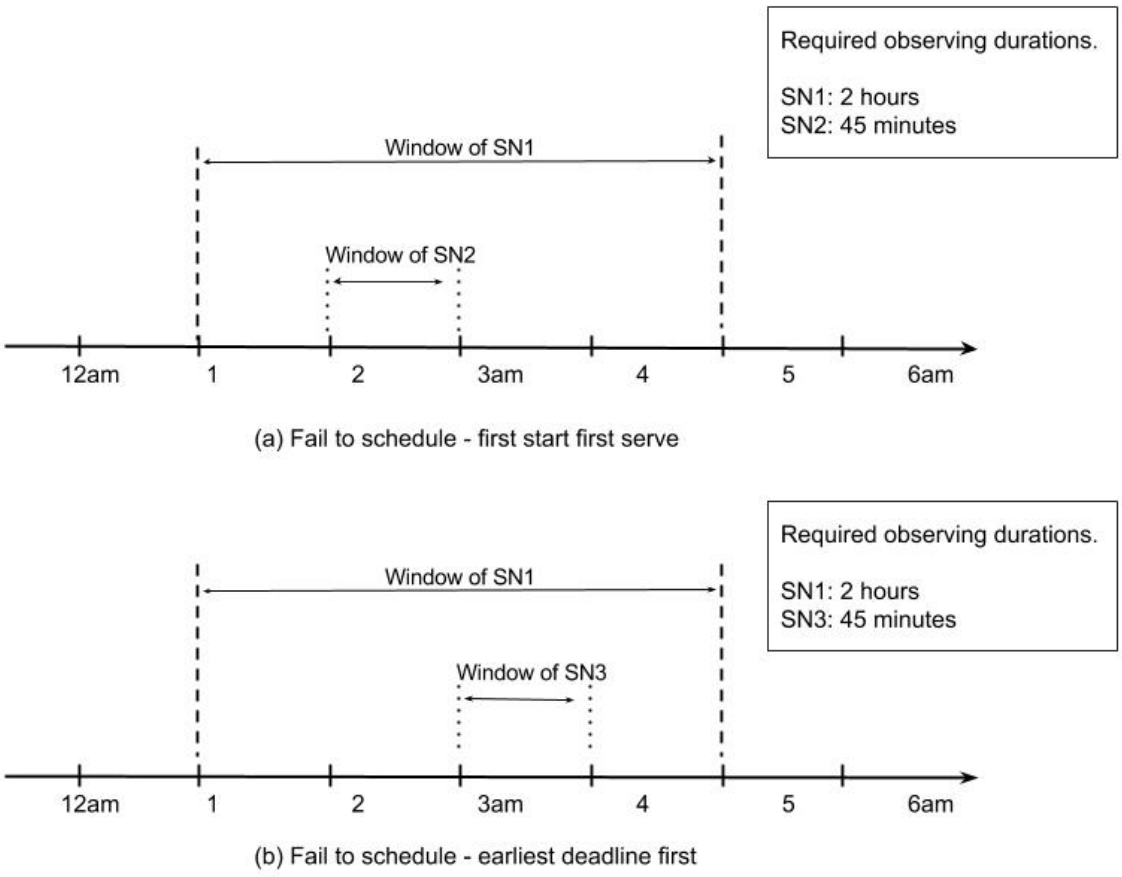


Figure 4.3.1 Two special cases of scheduling failure

4.3.2 Global Scheduler

The global scheduler on top of the local scheduler is responsible for providing the candidate list of objects (supernovae) for local scheduling in each night. As mentioned in section 4.2, our problem can be viewed in two different ways: locally (bin packing) and globally (scheduling tasks with soft deadlines). From the global scheduler's perspective, the underlying concerns (described below) drive us to model the problem in a different way in contrast with the local scheduler.

First, the fact is that the brightness of a supernova keeps decreasing after the supernova's peak brightness date (Figure 4.3.2), which affects the quality of observation. To avoid collecting low-quality data, the best practice is that (according to our astronomer collaborators):

1) Finish at least three observations between 0-20 days after peak brightness,

2) Only consider additional observations up to 100 days after peak as useful.

This adds a soft deadline for every supernova to be scheduled implicitly. For the second, as each observation consists of several observation instances (typically more than 3), it is possible that there is no room to schedule the last few instances for a certain supernova. In this case, such an observation is incomplete. Considering a plan that involves two supernovae A and B: A finished 4/5 and B finished 2/3 of the required number of observations. Both A and B are incomplete and thus are likely to be useless for further studies. A better plan could be 5/5 A and 1/3 B, or 3/5 A and 3/3 B, in which case at least one observation can be finished completely. Lastly, we should

note that the observation with the earliest deadline does not always need to be prioritized. We should also be aware of the actual workload left for the observation task.

In all, the global scheduler needs to make the decision globally when considering which observation tasks are more urgent to be scheduled for the current night. While it makes a global arrangement, the second-level scheduler figures out the details in the local scheduling process.

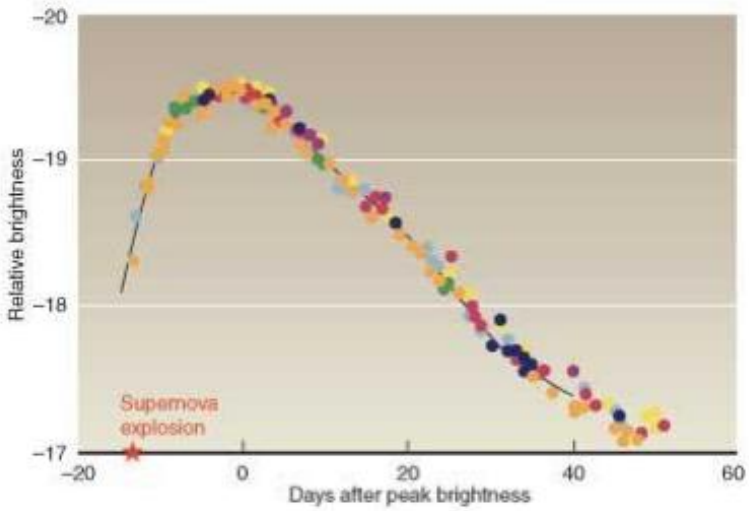


Figure 4.3.2 Supernova brightness decreases over time

4.3.2.1 EDF, SRTF, and LST

For the concerns discussed in the previous section, there are 3 classic scheduling algorithms that suit our problem perfectly: Earliest Deadline First (EDF), Shortest Remaining Time First (SRTF) [25], and Least Slack Time (LST) scheduling [26].

EDF or least time to go is a dynamic scheduling algorithm usually used in real-time operating systems for placing processes in a priority queue. Whenever a scheduling event occurs

(task finishes, new task released, etc), the queue is searched for the process that is closest to its deadline. That process is then scheduled to be the next one for execution. In our scenario, an observation task plays the role of a process and the limitation about the supernova brightness yields a soft deadline naturally. So we can have some priority score for ordering, if we evaluate available observations using EDF.

In **SRTF** scheduling, the process with the smallest amount of time remaining until completion is selected to execute. By definition, the current executing process is the one with the shortest amount of time remaining and the remaining time decreases as execution progresses, so the processes will always run until they complete or a new process which requires a smaller amount of time is added. Again, in our scenario, observation tasks stand for processes, and the number of remaining observation instances of a task is considered as "remaining time". Therefore, we have another way to evaluate the priority score of observations.

LST assigns priority based on the slack time of a process. Slack time is the amount of time left before a job's deadline after the job completes execution if the job was started as soon as it arrived. Namely, it is the temporal difference between the deadlines, the ready time and the run time. More formally, the slack time of a process is defined as (d - t) - c', where d is the process deadline, t is the real time since the job start, and c' is the remaining computation time. Back to our scenario, d will be a particular date after the peak brightness date, t will be the date of the current night under scheduling, and c' will be the number of remaining instances (because each instance can be finished in one night).

To conclude, we have proposed 3 different approaches for priority evaluation, focusing on different concerns in scheduling. In the next subsection, we combine them together to measure priority in a comprehensive manner.

4.3.2.2 Global Heuristic

Since we do not know which of the three policies (EDF, SRTF, LST) would work best for the different users, we combine them together and have different weights assigned to each one, to indicate how important each one is. So we propose the global heuristic in Equation 4.3.6:

$$H_{global}(SN) = w_5 * \frac{1}{deadline} + w_6 * \frac{1}{remaining\ work} + w_7 * \frac{1}{laxity} \quad \textbf{(Eq. 4.3.6)}$$

According to the heuristic, the observation of those supernovae that are further from their peak date and have few remaining instances plus less laxity to be postponed get higher priority scores. Similarly to our local heuristic, weights w5 - w7 are added as coefficients to tune the influence of each factor. In a broader view, our system collects user feedback and learns from past user feedback in order to tune these weights automatically.

4.3.2.3 Global Algorithm

Once the heuristic function is devised, the global algorithm is quite straightforward, as follows:

1. Iterate through a list of dates, which represents all the nights available in a current plan.

2. Get all the supernovae available for observation on a particular date.

3. Sort the available tasks in the order of priority, evaluated by H_{global}.

4. Get the top K supernovae in the sequence to be the candidate list; K is proportional to that night's capacity. Send the candidate list to the local scheduler for this night's local scheduling.

5. Repeat steps 2, 3, 4.

4.4 ADVANCED OBSERVATION SCHEDULING

The Advanced Observation Scheduling component (AOS) contains two main features. The first feature is calculating proper night capacity distribution for the current plan, making sure to reserve appropriate amount of "space" for upcoming new supernova. The reason is the potential for newly discovered supernovae in the near future, which could be more interesting or valuable to the astronomer, who is willing to reserve some "capacity" when making the plan for the current target list. The second feature is using a semi-supervised learning technique to tune the weights of factors considered in BOS, and adjusting the scheduling behavior for each user accordingly. As mentioned, a lot of factors can be considered for scheduling, but the potential weight of each factor (i.e., its importance) varies from user to user. For example, some astronomer may state that "at least 3 observation instances done within 20 days after the brightness peak of supernova" is necessary. In that case, the weight of the factor "soft deadline" is implicitly increased. In general, for the same target list, a different user would like to see a different observation plan, even with the exact same experiment configuration.

4.4.1 "Space" reserving Feature

This feature enforces an appropriate capacity distribution strategy and calculates how to assign the particular capacity for each night in the plan. The local scheduler will arrange the observation tasks based on such capacity each night, and the reserved capacity can be useful in the future. The main goal here is to try to reserve as much capacity as possible, without affecting the overall schedulability, which could lead to a failure in planning or an imperfect plan.

4.4.1.1 Plan with capacity variation

We know that the maximum amount of time that can be used for observations each night is already given by the user, but it does not mean that all of the time needs to be pre-planned. There can be a reserved part that can be spent in the future. A typical case is that the user at first requests a plan be generated for his/her current targeting list. However, later on, messages/reports about new detected supernovae are released and captured by our system. Then the user hopes to add more objects into the targeting list while the current plan is being executed. With the purposefully reserved capacity, the additional objects deserve a bigger chance to be scheduled as well. Different users may prefer different strategies to reserve such space capacity. The next subsection discusses this in more detail.

4.4.1.2 Three alternatives for capacity distribution

Currently, the front-end of our system provides three different strategies for a user to choose to distribute capacity. The user needs to understand the pros and cons of each strategy, making his/her own choice. The strategies are as follows:

Normal This is the most natural strategy: putting all the capacity into the current planning process. Whatever capacity left after usage will be the reserved capacity for the future. In this case, the observations for objects in the current targeting list would be finished as soon as possible, even if some of them are not that urgent. Generally, the capacity distribution under this strategy would be similar to that shown in Figure 4.4.1.

Evenly If an observation needs k amount of capacity and should be finished in n days, we add k/n capacity usage evenly into those n days. By doing so for all the current observations, the

cumulative capacity usage as a whole would be roughly evenly distributed. With some increasing adjustments, such capacity distribution would not affect the scheduling, although there is no such guarantee. The capacity distribution curve is shown in Figure 4.4.2.

Inversed The main idea of this strategy is to delay current observation tasks as long as it does not affect their schedulability. For example, assume an observation task needs c capacity for each of its instances, the total observation time is 4, and the gap is 1 day, the two deadlines for it are d_1 and d_2. In this way, c amount of capacity usage should be added into days $d_1 - 4$, $d_1 - 2$, d_1, d_2. The details about the calculation will be described in the next subsection. The main point is to find space at the early part of the available nights for coming observations. The idea is illustrated in Figure 4.4.3.

Suppose there is a group of interesting supernovae coming soon in the current plan's time interval. If their active windows are concentrated in the back portion of the current time interval, strategy 1 would be a good fit. But if they are very active at the front portion of the current time interval, strategy 3 should be used to handle such a case. At last, if they distributed sparsely in the current time interval, strategy 2 would be the most effective way to deal with that. Clearly, there is uncertainty and no strategy is a clear winner for all cases, but the user can have a specific preference. If a user considers the upcoming supernovae much more valuable than all current ones, he/she could always choose to stick on inversed capacity distribution strategy under whatever circumstance.

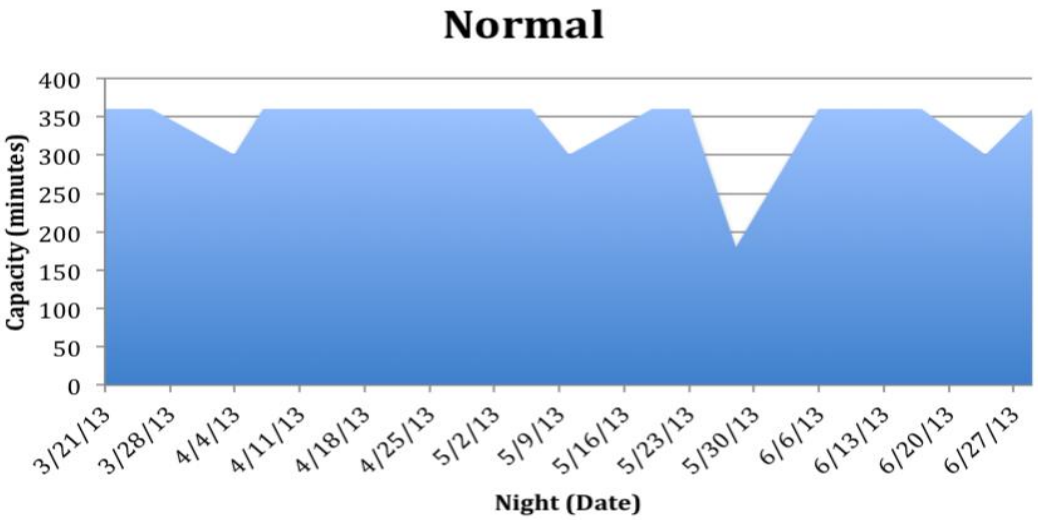


Figure 4.4.1 "Normal" night capacity distribution in certain plan

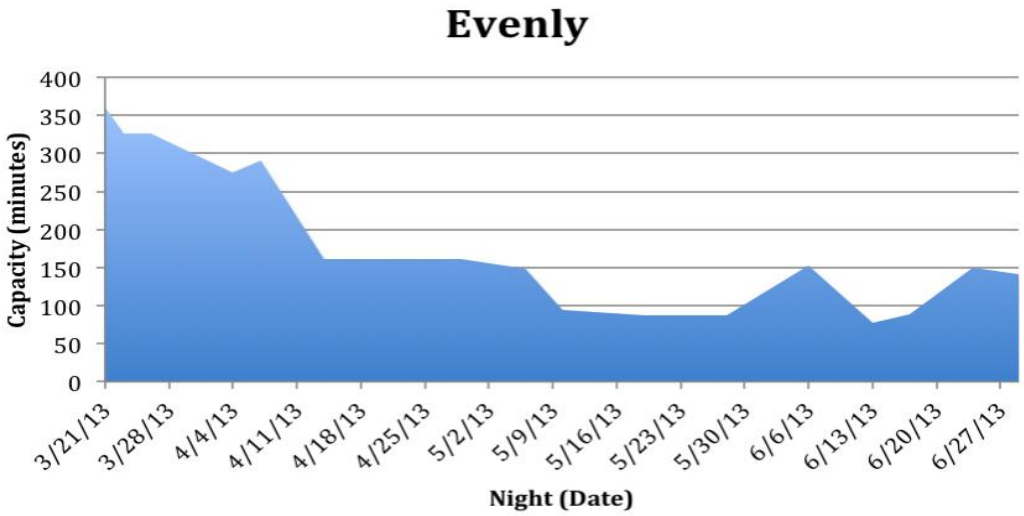


Figure 4.4.2 "Evenly" night capacity distribution in certain plan

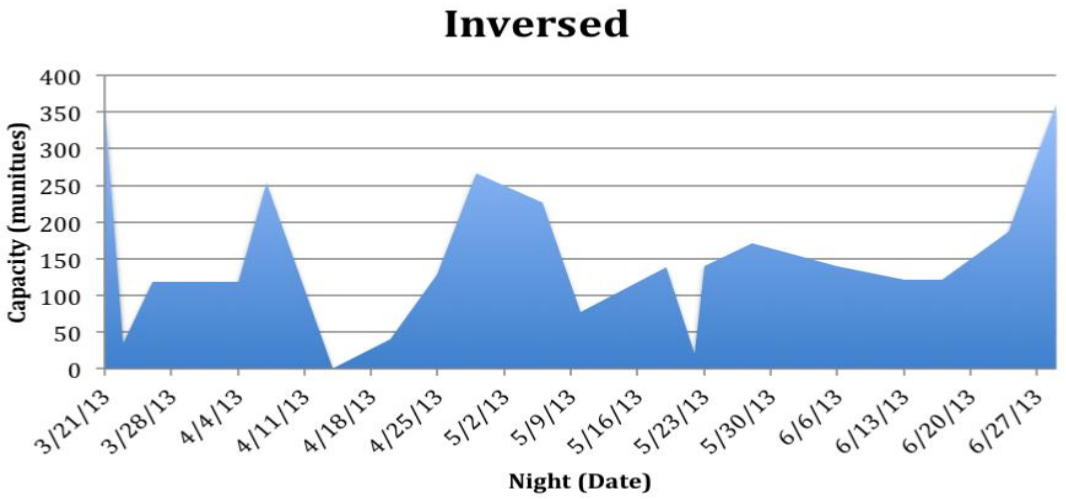


Figure 4.4.3 "Inversed" night capacity distribution in certain plan

4.4.1.3 Generate valid capacity distribution

It is trivial to enforce the normal distribution strategy. For the other two, the valid capacity distribution is generated by iterating every object (observation) and accumulating each object's capacity usage. The algorithms for them are too detailed to include, but the main steps are as follows:

Evenly strategy:

1) Iterate through every object (observation),

1.1) Add average capacity usage into nights between the plan start date and deadline 1, take the first 3 instances' capacity consumption into account.

1.2) Add average capacity usage into nights between the plan start date and deadline 2, take the remaining instances' capacity consumption into account.

2) Increase the cumulative capacity usage by a certain percentage (adjustment) to get the final capacity distribution. We used a percentage of 10% in our experiments.

Inversed strategy:

1) Iterate through every object observation

*1.1) Do backward traverse from deadline_1, adding instance capacity usage into nights [deadline_1 – 2*gap – 2, deadline_1 – gap – 1, deadline_1] (if instance number > 3).*

1.2) Do backward traverse from deadline_2, adding instance capacity usage into nights starting from deadline_2, every other k nights (k is the observation gap).

2) Increase the cumulative capacity usage appropriately, and return the final capacity distribution. Again, we used a percentage of 10% in our experiments.

4.4.2 Semi-supervised learning feature

As mentioned earlier, for both the local and the global scheduler in BOS, there is a weight parameter associated with each factor which could affect the planning. This is because even though we know which ones are important factors, we are not exactly sure about how important each factor is to each user. So we define a set of weights, $w_1 \sim w_7$, each representing how important the corresponding factor in the local and the global heuristics is. Each weight has a fixed and unique range of values. Instead of guessing or using a global set of values for these weights, we rely on each user to give us feedback and learn his or her preferences. The basic idea is: first we let the user evaluate and provide feedback for each plan, and then we record the user's feedback with different configured set of weights. We use the feedback and the corresponding sets of weights to build the training set. If the training set is not well-built, we

assign arbitrary values for each weight; if the training set has been built, we learn the next set of weights from the training set.

In the following sections, we will introduce how to learn from history data, how to build our training set, and how to generate plans that users consider preferences by tuning weights iteratively.

4.4.2.1 Learning with logistic regression

In statistics, logistic regression [27, 28] is a type of regression analysis used for predicting the outcome of a categorical dependent variable based on one or more predictor variables. Typically, "logistic regression" is used to refer specifically to the problem in which the dependent variable is binary. (For our problem, the dependent variable is the feedback from the user, which is either positive or negative, and the predictor variables are the weights of different factors.) In binary logistic regression, the outcome is usually coded as "0" or "1. For our task, if a plan generated by BOS given a specific configuration of weights is considered as positive, it is coded as "1"; on the contrary, it is coded as "0". So the logistic regression here is used to predict the probability of a plan being positive given a set of predictors (i.e., the weights of the factors in BOS).

4.4.2.2 Building the training set

The logistic function needs a training set, consisting a bunch of instances, to train the model. In our problem, each instance in the training set is represented as:

[label, feature]

Label is the dependent variable. It is 1 or 0 for positive or negative feedback provided by users. *Feature* is the set of predictor variables. Each predictor is the weight used in BOS. A *feature* is denoted as:

{w1:[0,30], w2:[0,30], w3:[0,30], w4:[0,30], w5:(0,10), w6:(0,10), w7:(0,10)}

The algorithm used to train the model and predict the probability is shown in Algorithm 4.4.1.

Algorithm 4.4.1

```
Func train(dataList, n):
    // n is the number of training iterations.
    For i from 0 to n:
        For [label, feature] in dataList:
            predicted = classify(feature)
            // f is feature, v is its value
            For f, v in feature.items():
                If f is not existing weight:
                    W[f] = 0
                update = (label - predicted) * v
                W[f] += bias * update
    Return
```

```
Func classify(feature):
    // logit is the inverse of logistic function
    logit = 0
    For f, v in feature.items():
        // ceof is coefficient
        ceof = 0
        If f is existing weight:
            coef = W[f]
        logit += coef * v
    // exp return e raised to specified power
    P = 1.0 / (1.0 + math.exp(-logit))
    Return P
```

4.4.2.3 Interacting with users

A specific training set is maintained for each user, and it grows when the user provides a feedback for a plan. The user can decide when to allow learning from the training set, and typically the bar is higher than a default threshold defined by the system. One thing to notice is that, the set of weights of a plan with positive feedback from a user may not be directly reused

for generating the next plan; instead, the weights associated with that plan will be added into the training set. For the same input factors, when the user asks for another plan suggestion, the trained model will adopt the best set of weights learned from the training set, including the new positive feedback plans. This is because even if the user thinks the previous plan is a successful or satisfactory one, it cannot guarantee it is the most optimal one. So when a user interacts with the same input set up, our system is trying to be aggressive – providing more reasonable plans, collecting more feedback from the user. On the other hand, if the user inquiries for a plan with totally different input factors, our system would like to provide the best ranked set of weights to the BOS to generate a plan, unless the user explicitly lets the system to start learning from scratch. From this perspective, the training set is also equivalent to a historic database, and our system is able to provide an interface for the user to give a "satisfaction" score to plans and subsequently learn the user's own preferences.

4.5 EXPERIMENTAL ANALYSIS

Experiments were run on the "Elements" Cluster (elements.cs.pitt.edu) in the Computer Science department, of the University of Pittsburgh. The configuration of the machines in the cluster is: Dual Hyper-Threaded Six-Core 3.33GHz Xeon processors, 96GB RAM memory, 64-bit Linux architecture, running CentOS 5.5 with kernel version 2.6.

4.5.1 Analysis of Local Algorithms

In section 4.3.1, three different local algorithms and one baseline algorithm were presented. The following experiments evaluate their performance based on two metrics - response time and profit gain. For the local heuristic shared by the algorithms, their associated weights are set to the equal default value (e.g., 1).

Before the experiments, we need to tune the parameters in each algorithm to achieve its best performance. The relevant parameters of algorithm 1 (greedy beam search), algorithm 2 (iterative dynamic programming), and algorithm 3 (random restart hill climbing) are, beam_width, iteration_time, and restart_num, respectively. We define three cutoff ratios to quantify those variables and relate them with the candidate list size:

$$beam\_width = candidate\_list\_size * cutoff\_ratio1,$$

$$iteration\_time = candidate\_list\_size * cutoff\_ratio2,$$

$$restart\_number = candidate\_list\_size * cutoff\_ratio.$$

Algo 1 - cutoff ratio 1	Response Time	Accuracy
0.1	0.026	5.141
0.15	0.039	5.203
0.2	0.051	5.301
0.25	0.068	5.299
0.3	0.081	5.425
Algo 2 - cutoff ratio 2	Response Time	Accuracy
0.3	0.011	6.081
0.4	0.012	6.070
0.5	0.011	6.082
0.6	0.012	6.085
0.7	0.012	6.073
Algo 3 - cutoff ratio 3	Response Time	Accuracy
0.65	0.072	6.194
0.7	0.087	6.206
0.75	0.102	6.239
0.8	0.116	6.237
0.85	0.131	6.242

Table 4.5.1 Tuning cutoff ratio for the three algorithms

Table 4.5.1 shows performance differences when tuning cutoff ratios. We can see that for all three algorithms, the higher the cutoff ratio, the more profit gain. The reason is that the algorithms become closer to optimal, as they are closer to an exhaustive search and less to a version using just heuristics. The trade-off is that the response time increases at the same time. In order to keep a good balance, cutoff-ratio 1, 2, and 3 are set to 0.2, 0.5, and 0.75 respectively in the following experiments.

Figure 4.5.1 and Figure 4.5.2 show the results with the above cutoff-ratio values. The x-axis denotes the ratio of candidate list capacity consumption sum to assigned night capacity. First

of all, all three algorithms' response time is dramatically reduced compared to the baseline algorithm (i.e., the brute-force algorithm), while there is little profit loss. For the comparison among the three algorithms, *Algo2* performs best regarding response time, while *Algo3* performs worst. *Algo1* gains the least profit of all, while the other two algorithms have very similar performance (*Algo3* gains slightly more than *Algo2* for overall). Thus, the iterative dynamic programming approach is the best one in our case.

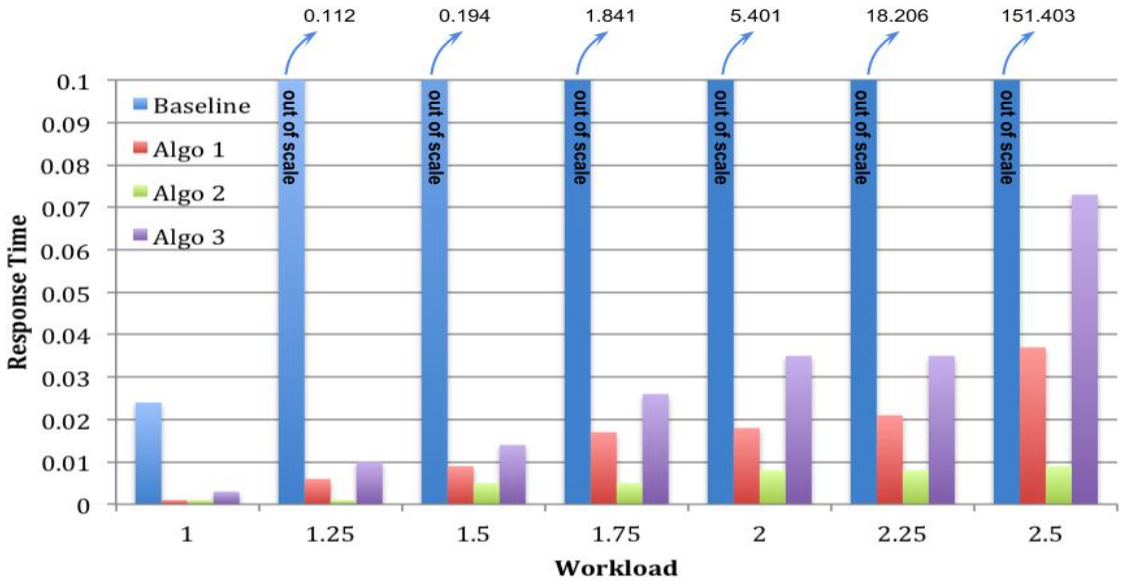


Figure 4.5.1 Comparison of response time

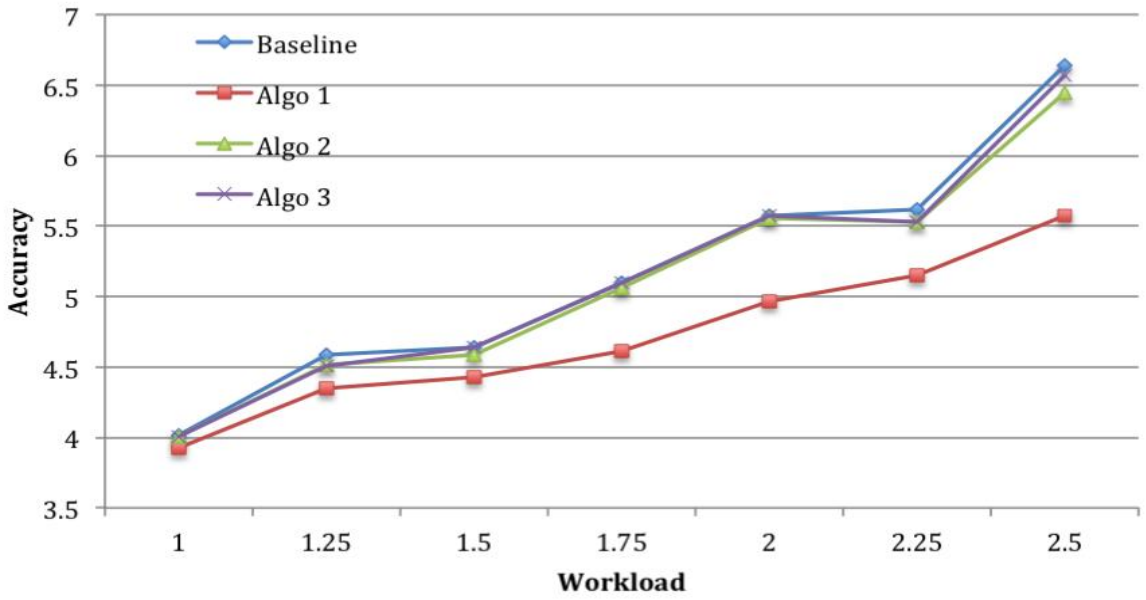


Figure 4.5.2 Comparison of accuracy

4.5.2 Effect of Learning for local/global Scheduling

To prove the assumption that the semi-supervised learning technique embedded in the planner can influence the result plan suggestion, and improve user's satisfaction with the process of interaction and iteration, we conducted the following experiments.

The first experiment shows that the learning technique helps improve local scheduling. Assuming a user submitted a request to plan for 3 imaginary supernovae: 2014A, 2014B, and 2014C. The detailed input information is as follows:

2014A – B-Peak: 2014-03-21, Priority: 0.9, obsDuration: 45, obsTimes: 1, obsGap: 1

2014B – B-Peak: 2014-03-21, Priority: 0.5, obsDuration: 25, obsTimes: 1, obsGap: 1

2014C – B-Peak: 2014-03-21, Priority: 0.5, obsDuration: 25, obsTimes: 1, obsGap: 1

Nights in the plan: 2014-03-21, 2014-03-23

Hours per night: 1

(All the remaining data related to the 3 supernovae are omitted for brevity.)

There are two ways to plan for the request: i) schedule the observation of 2014B and 2014C on the first night, then observe 2014A on the second night, and ii) schedule the observation of 2014A on the first night, and use the second night for the observation of the other two supernovae. Obviously, the way to observe 2014B and 2014C on the first night is better than the alternative. The reason for that is because the profit gain is more by scheduling 2014B and 2014C on 2014-03-21, which is the peak brightness date for all three supernovae. A driver program was written to simulate user actions. Every time it receives the better plan suggestion it gives positive feedback. Otherwise it gives negative feedback. Figure 4.5.3 (a) shows the percentage of good plan suggestions with different training set sizes.

The second experiment demonstrates the influence on global scheduling. The devised scenario is following:

2014D – B-Peak: 2014-03-03, Priority: 0.9, obsDuration: 50, obsTimes: 2, obsGap: 1

2014E – B-Peak: 2014-03-03, Priority: 0.9, obsDuration: 50, obsTimes: 1, obsGap: 1

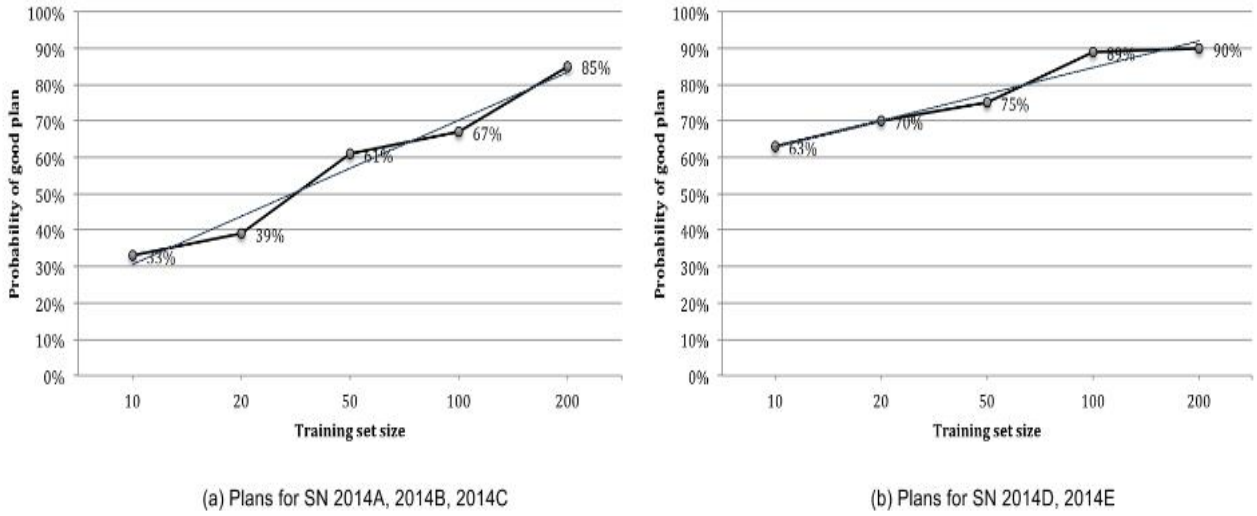
Nights in the plan: 2014-03-21, 2014-03-22, 2014-03-23

Hours per night: 1

(All the remaining data related to the 3 supernovae are omitted for brevity.)

We can notice that 2014-03-23 is the deadline for the observation of both supernovae. And if the planner chooses 2014E at the first night, the observation of 2014D cannot be finished in two

continuous nights. So a good plan suggestion should arrange 2014D on the first and last night, using the night in the middle to observe 2014E. As the deadline is the same and 2014E has less unfinished workload (observation instances), the only way to make global heuristic evaluate 2014D more urgently than 2014E is to overweight the third factor – slack time. Figure 4.5.3 (b) illustrates the behavior of our system using this approach.



(a) Plans for SN 2014A, 2014B, 2014C

(b) Plans for SN 2014D, 2014E

Figure 4.5.3 Learning affecting decision making

4.5.3 Overall Performance Analysis

Finally, we provide the automatically generated plans by our system, comparing to the manually generated plan by our astronomer colleagues for *list spring 2013* (see **APPENDIX B**). Our colleagues from Department of Physics & Astronomy inspected our plan results, admitting the

feasibility of those plan suggestions, even though the generated plans were not the same with what the astronomers manually planned.

5.0 CONCLUSIONS AND FUTURE WORK

5.1 SUMMARY OF CONTRIBUTIONS

Supernovae researchers need to track lists of interesting supernovae and create observation plans. When they are looking at the dates allocated to them for observation and are trying to determine which supernovae to observe in which date, considerations about the temporal constraints and the potential conditions on limited time and resources overwhelm the astronomers, and incur difficulties in generating efficient observation plans. In this work, we proposed the SNeT (computer-assisted SuperNovae Tracking) system for addressing the problems supernovae researchers encounter when making observation plans.

Specifically, this thesis made the following algorithmic and system contributions:

- A *Supernova Data Aggregator* (SDA), that integrates existing supernova messages/reports published by different sources and builds a local supernova data knowledge-base.

- A *Supernova Tracking Management System* (STMS), that enables the user to manage the entire supernova tracking process – (1) searching for interesting supernova targets, (2) creating a potential list of supernovae for observation, (3) configuring the input parameters of a plan, and (4) displaying and recording generated observation plans in a user-friendly way.

- A *Supernova Observation Plan Generator* (SOPG), that consists of Basic Observation Scheduling (BOS) and Advanced Observation Scheduling (AOS) modules. SOPG generates feasible and efficient observation plans under the comprehensive consideration of all possible factors that affect them.

The SOPG component as the primary contribution of this thesis, it addresses the most challenging problem. The SOPG component can also be thought as an independent system of scheduling, working for SNeT. SOPG is structured in two layers, namely BOS and AOS, which work collaboratively in generation good quality observation plans that meet the users requirements and preferences.

The BOS module of SOPG breaks down a complex scheduling problem and solves it in two phases. The global scheduler considers observation tasks in a broader view and orders the tasks according to both their deadline and progress. On the other hand, the local scheduler narrows down the problem and arranges tasks within a single night. We developed three different algorithms to optimize the local scheduler: "*greedy beam search*", "*iterative dynamic programming*", and "*random restart hill climbing*", based on three different ideas - greedy approximation, dynamic programming, and local search (typically used for optimization problems in AI), respectively. Our experiments suggest that the above three algorithms achieve a balance in efficiency and quality.

The AOS module of SOPG extends and enhances the system in two ways. First, the "space" reserving feature enables users to choose from three flavors of night capacity distribution. The way night capacity is assigned (or reserved) impacts the capability for

observation of supernovae later determined as interesting and valuable but not included in the original set of supernovae to be observed. In other words, it enables to dynamically include in an existing observation plan new supernovae observation as well as to repeat failed ones.

The second feature of AOS is the semi-supervised learning embedded in our system. In BOS module, we devised heuristic functions for decision making in both global and local schedulers. These heuristics combine all factors affecting the scheduling and attach adjustable weights on them. Thus, users can interact with SOPG and adjust plans by changing the weights in the heuristic functions. Basically, the system records users' opinions, builds and maintains training sets, which would be used to learn the appropriate weights. As the weights in the heuristic functions are being tuned implicitly, SOPG provides customized results to users. As demonstrated in the evaluation part, this feature can help our system achieve a better solution without affecting the applicability of the underlying algorithms.

5.2 FUTURE WORK

Although the work we have completed indeed solves our astronomer collaborator's basic problem, there are several areas also worth further investigations, as follows:

- For SDA: A customizable data downloading and parsing engine. It should allow users to add their own sources, and register their own filtering rules (to parse and extract useful data). Also useful would be an interface for users to analyze supernova messages/reports manually and contribute their data to the supernova data knowledge-base. Both features can improve SDA as the supplement of default data collecting routine.

- For STMS: Several improvements can be made on "experiment configuration" and "plan management" features. For the former, a well-designed access control system enabling users to either protect or share their annotation, opinion, and progress on supernova observations is needed. For the latter, a refined UI visualizing detailed statistics about the on-going observation plan would be helpful.

- For SOPG: Enhancing the system with dynamic rescheduling. Basically, users can interact with the system and inspect the on-going plan and current observation status. This would enable users to do the following: 1) add/remove objects and constraints, 2) change object's attributes or corresponding constraints, 3) cross out certain night or certain period of time in one night as failures (device broken down, bad weather, etc). As a result, the enhanced system can combine the changes with the initial configuration to reschedule the remaining portion of the whole observation.

5.3 FINAL THOUGHTS

SNeT provided a whole solution package for the astronomers observing supernovae. In the process of designing and developing SNeT, we also studied and explored interesting scheduling and optimization problems. We designed and implemented algorithms to address them. We consider our work as an early step in the exploration of building intelligent and easy-to-use supernovae tracking systems. We hope our work can help the research of astronomers and computer scientists interested in this area.

Acknowledgements: We first want to thank Professors Michael Wood-Vasey and Jeffery Newman, our Astronomer collaborators. We also like to thank Nikhil Venkatesh and Wen Gao from the ADMT Lab for the help of implementing part of STMS and the web-UI integrated into AstroShelf platform. This work was funded in part by NSF award OIA-1028162.

APPENDIX A

STRUCTURE OF SN CLASS

A very important part for the implementation of the planning system is the "SN class", which encapsulates all the information of the basic entity (supernova object) enrolled in the process of plan generation. As mentioned in section 4.1, user input contains a list of objects of interest with a list of observation constraints, which indicates the detailed requirements for the observation of certain supernova. So, the object's data associated with its constraints provide all the useful information about each object for planning. Considering that, we designed and implemented the "SN class" to integrate this information together, providing standardized interfaces to access them, recording additional status data for the object, and giving useful helper functions to manipulate the object.

The variables and methods integrated in SN class are listed below.

Variables:

1. Catalog data of the object (supernova):

- SN.Name: the name of supernova, e.g., SN 2013A, PSN J01340066-3423404, MASTER OT J111154.50+453214.9, CSS130828:223849-251246, etc.

- SN.RA: the right ascension to measure the position of supernova.

- SN.Dec: the declination to measure the position of supernova.

- SN.Type: the classification of supernova, divided into Type I and Type II.

- SN.Redshift: the redshift of supernova.

- SN.Mag: the apparent magnitude of supernova, to measure brightness.

- SN.BPeak: the date when supernova in its peak brightness.

- SN.Peakdate: the language built-in datetime object converted from date string.

- SN.Priority: a number in range (0, 1) assigned by user, to measure how valuable the success in such supernova's observation is. Typically, the priority is very important fact to consider in planning. For a feasible observation plan with no deadline misses, no time conflict, etc, the maximum gain in priority raise a positive flag for user's acceptance of the plan.

2. Constraints for the observation of the object (supernova):

- SN.obsDuration: it measures how long (in minutes) each observation instance of a particular object should takes. The time varies for different night.

- SN.obsGap: it gives a constraint that how long (in nights) two adjacent observation instances of the same supernova should be separated.

- SN.obsTimes: it measures how many instances are needed for such supernova observation to complete.

3. Status data of the object (supernova):

- SN.status.last_instance: the index of the last finished instance.

- SN.status.last_time: the index of night in which the last instance finished.

- SN.status.remaining_instance: number of remaining instances to be schedule.

- SN.status.miss_deadline: flag to mark whether observation of such supernova passes the deadline or not.

Methods:

1. Methods to access variables:

- SN.getPriority(): return the priority of such supernova given by user.

- SN.getObsDuration(): return the observation duration of such supernova according to the date (of the night) information.

- SN.getObsGap(): return the observation gap set up by user.

- SN.getObsTimes(): return the observation times set up by user.

- SN.update_status(): it will update the recorded status as new instance finished.

- SN.reset_status(): it will reset the status of observation to initial, as user may iterate through multiple plans with the same input parameters.

2. Helper functions:

- SN.available(): given a particular night, the function will check whether it is permitted or not to schedule the observation of such object, considering all the existing constraints.

- SN.get_deadline(): given a particular night, return the relative deadline (in unit of nights) for finishing the observation of such object.

- SN.get_remaining_work(): return the amount of unfinished workload (in unit of minutes) of particular object, by adding all the remaining instances' duration.

- SN.get_slack(): given a particular night, return the amount of time (in unit of nights) left after a job (observation) if was started immediately.

- SN.get_obsWindow(): calculate the specific time window for observation in particular night, e.g., start at 12:30 am, end at 1:45 am. The calculation is based on R.A., Dec., and the date of that night.

When the planning system accepts the information of target list and constraints from user, it will instantiate an object of "SN class" stated above, for each supernova object needs to be tracked in current plan. A list of such SN class objects is the core data structure for performing scheduling algorithms. For example, global scheduler in BOS iterates over a list of SN class objects, use their helper function to check the availability, to calculate the value of factors affecting scheduling; Local scheduler in BOS iterates over the candidate list from the global one, choosing the best portion of it as tonight's tasks, updating their status, etc. More details will be disclosed in following sections.

In all, the implementation of SN class plays an important role for building the whole planning system, as it makes the tasks of schedulers built upon it easier, by grouping useful data, keeping intermediate status, and providing helper functions.

APPENDIX B

PLANS FOR *LIST SPRING 2013*

In the following pages we list the supernovae observation plan generated by our astronomer colleagues along with three plans automatically generated by SNeT. Although these three plans are not identical to the manually generated one, they were still deemed reasonable by our astronomer collaborators.

[Manual plan by Dr. Michael Wood-Vasey for observations in spring 2013]

Name | z | B-Peak | 20130321 | 20130326 | 20130420 | 20130425 | 20130429 | 20130519 | 20130522 | 20130523 | 20130528 | 20130613 | 20130617 | Notes

2012cg | 0.001458| 20120401 | ..,..,.. | ..,..,.. | ..,..,.. | 16,25,.. | ..,..,.. | ..,..,.. | ..,..,.. | 16,25,.. | 25,34,.. | ..,25,.. | 16,25,.. | CBET 3111
2013cs | 0.009240| 20130524 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 09,09,09 | 09,09,09 | 09,09,09 | 09,09,09 | CBET 3533 LSQ13aiz ATEL 5067
2013da | 0.0216 | 20130606 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 09,09,09 | 09,09,06 | Observed with Super-LOTIS
2013bs | 0.0276 | 20130421 | ..,..,.. | ..,..,.. | 09,16,.. | 15,15,.. | ..,28,.. | ..,..,.. | 16,17,.. | ..,..,.. | 16,16,.. | 25,16,.. | 25,25,.. | ATel 4993, CBET 3494 PSNJ17172203+4104002
PTF13asv | 0.035 | 20130511 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 25,25,.. | ..,..,.. | 25,25,.. | 25,25,.. | 16,16,.. | ATEL 5061
2013bo | 0.036 | 20130411 | ..,..,.. | ..,..,.. | 09,16,.. | 25,16,.. | 16,16,.. | ..,..,.. | 25,25,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ATel 4989 CSS130415:131729+424430, CBET 3490
2013bt | 0.0364 | 20130426 | ..,..,.. | ..,..,.. | 16,16,.. | 09,16,.. | 09,16,.. | 25,25,.. | ..,..,.. | 25,35,.. | ..,..,.. | ..,..,.. | ..,..,.. | 4993, PSNJ14211513+6134159 CBET 3497
SNhunt175 | 0.0409 | 20130318 | ..,50,.. | ..,..,.. | 20,16,.. | ..,..,.. | ..,41,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ATel 4896
2013ck | 0.049 | 20130512 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 16,25,.. | ..,..,.. | 34,41,.. | ..,..,.. | 25,25,.. | 25,25,.. | CBET 3523
CSS130218:092354+385837 | 0.05 | 20130310 | ..,..,.. | ..,25,.. | ..,41,.. | ..,25,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ATel 4908
PTF13ayw | 0.0538 | 20130518 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 16,36,.. | ..,..,.. | ..,..,.. | 25,41,.. | 25,25,.. | 25,16,.. | ATEL 5061
2013cb | 0.0541 | 20130512 | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | 25,25,.. | ..,..,.. | 25,41,.. | ..,41,.. | ..,..,.. | ..,..,.. | CBET 3509
2013bq | 0.06 | 20130424 | ..,..,.. | ..,..,.. | 25,25,.. | 25,25,.. | 16,25,.. | ..,..,.. | ..,..,.. | ..,50,.. | ..,..,.. | ..,..,.. | ..,..,.. | ATel 4993, CBET 3492 CSS130415:130408+435408
2013ar | 0.06 | 20130323 | ..,..,.. | 25,41,.. | 25,25,.. | ..,..,.. | 41,41,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | CBET 3446
2013be | 0.065846| 20130416 | ..,..,.. | ..,..,.. | ..,..,.. | 25,25,.. | 41,50,.. | 41,41,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | CBET 3470, iPTF13aig ATEL 5019
CSS130317:082848+293031 | 0.08 | 20130319 | ..,..,.. | 41,41,.. | 41,36,.. | 41,50,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ..,..,.. | ATel 4984

[Plan 1 generated by SOPG, SNeT]

##
2013-03-21: 2013cb, 2013cs, CSS130317:082848+293031, 2013bt, CSS130218:092354+385837, 2013bo, SNhunt175, 2013bs
2013-03-26: CSS130218:092354+385837, 2013bo, 2013ar, 2013bt, 2013cb, 2013da, SNhunt175, 2013bs, 2013cs
2013-04-20: 2013bo, SNhunt175, 2013bq, 2012cg, 2013bt, PTF13asv, 2013ar, CSS130218:092354+385837
2013-04-25: PTF13asv, 2013bs, 2012cg, 2013ar, 2013bt, 2013cs, 2013bo, 2013bq
2013-04-29: 2013bs, 2013bt, 2013bq, 2013da, 2013cs, PTF13asv, 2013be, 2013cb
2013-05-19: CSS130317:082848+293031, 2013ck, 2013bq, 2013bs, PTF13asv, 2013be
2013-05-22: 2012cg, CSS130317:082848+293031, 2013bs, 2013ck, PTF13ayw
2013-05-23: 2013be
2013-05-28: 2012cg, 2013bs, 2013ck, PTF13ayw
2013-06-13: PTF13ayw, 2012cg, 2013ck
2013-06-17: PTF13ayw
##

##

Name	z	B-Peak	2013-03-21	2013-03-26	2013-04-20	2013-04-25	2013-04-29	2013-05-19	2013-05-22	2013-05-23	2013-05-28	2013-06-13	2013-06-17
2012cg	0.001	2012-04-01	-	-	41	41	-	-	41	-	41	41	-
2013ar	0.06	2013-03-23	-	65	65	65	-	-	-	-	-	-	-
2013be	0.065	2013-04-16	-	-	-	-	77	77	-	77	-	-	-
2013bo	0.036	2013-04-11	35	35	35	35	-	-	-	-	-	-	-
2013bq	0.06	2013-04-24	-	-	50	50	50	50	-	-	-	-	-
2013bs	0.027	2013-04-21	38	38	-	38	38	38	38	-	38	-	-
2013bt	0.036	2013-04-26	32	32	32	32	32	-	-	-	-	-	-
2013cb	0.054	2013-05-12	47	47	-	-	47	-	-	-	-	-	-
2013ck	0.049	2013-05-12	-	-	-	-	-	58	58	-	58	58	-
2013cs	0.009	2013-05-24	27	27	-	27	27	-	-	-	-	-	-

2013da	0.021	2013-06-06	-	27	-	-	27	-	-	-
-	-	-								
CSS130218:092354+385837	0.05	2013-03-10	30	30	30	-	-	-		
-	-	-	-	-						
CSS130317:082848+293031	0.08	2013-03-19	82	-	-	-	-	82		
82	-	-	-	-						
PTF13asv	0.035	2013-05-11	-	-	50	50	50	50	-	-
-	-	-								
PTF13ayw	0.053	2013-05-18	-	-	-	-	-	-	53	-
53	53	53								
SNhunt175	0.0409	2013-03-18	43	43	43	-	-	-	-	-
-	-	-								

##

##
7 SN Observations missed deadline, listed as following:

2012cg, 2013ar, 2013be, CSS130218:092354+385837, CSS130317:082848+293031, PTF13ayw, SNhunt175

0 SN Observation didn't finish completely, listed as following:

N/A

56.25% SN Observations fully scheduled on time!
##

[Plan 2 generated by SOPG, SNeT]

##
2013-03-21: 2013ar, CSS130218:092354+385837, SNhunt175, 2013bs, PTF13asv, 2013bq, 2013bo, 2013bt
2013-03-26: 2013ar, CSS130218:092354+385837, SNhunt175, 2013bs, 2013cb, 2013bq, 2013bo, 2013bt
2013-04-20: 2013ar, CSS130218:092354+385837, SNhunt175, 2013bs, PTF13asv, 2013bq, 2013bo, 2013bt
2013-04-25: 2012cg, 2013bq, 2013cb, 2013ck, 2013bo, 2013bt, PTF13asv, 2013bs
2013-04-29: CSS130317:082848+293031, 2013cb, 2013bt, PTF13asv, 2013bs, 2013da, 2013cs, PTF13ayw
2013-05-19: CSS130317:082848+293031, 2013be, 2012cg, 2013bs, 2013da, 2013cs, PTF13ayw
2013-05-22: CSS130317:082848+293031, 2013be, 2012cg, 2013bs, 2013cs, PTF13ayw
2013-05-23: 2013ck
2013-05-28: 2013be, 2012cg, 2013ck, PTF13ayw, 2013cs
2013-06-13: 2012cg, 2013ck
2013-06-17: Null
##

##

Name	z	B-Peak	2013-03-21	2013-03-26	2013-04-20	2013-04-25	2013-04-29	2013-05-19	2013-05-22	2013-05-23	2013-05-28	2013-06-13	2013-06-17
2012cg	0.001	2012-04-01	-	-	-	41	-	41	41	-	41	41	-
2013ar	0.06	2013-03-23	65	65	65	-	-	-	-	-	-	-	-
2013be	0.065	2013-04-16	-	-	-	-	-	77	77	-	77	-	-
2013bo	0.036	2013-04-11	35	35	35	35	-	-	-	-	-	-	-
2013bq	0.06	2013-04-24	50	50	50	50	-	-	-	-	-	-	-
2013bs	0.027	2013-04-21	38	38	38	38	38	38	38	-	-	-	-
2013bt	0.036	2013-04-26	32	32	32	32	32	-	-	-	-	-	-
2013cb	0.054	2013-05-12	-	47	-	47	47	-	-	-	-	-	-
2013ck	0.049	2013-05-12	-	-	-	58	-	-	-	58	58	58	-

2013cs	0.009	2013-05-24	-	-	-	-	27	27	27	-
27	-	-								
2013da	0.021	2013-06-06	-	-	-	-	27	27	-	-
-	-	-								
CSS130218:092354+385837	0.05	2013-03-10	30	30	30	-	-	-		
-	-	-	-	-						
CSS130317:082848+293031	0.08	2013-03-19	-	-	-	-	82	82		
82	-	-	-	-						
PTF13asv	0.035	2013-05-11	50	-	50	50	50	-	-	-
-	-	-								
PTF13ayw	0.053	2013-05-18	-	-	-	-	53	53	53	-
53	-	-								
SNhunt175	0.0409	2013-03-18	43	43	43	-	-	-	-	-
-	-	-								

##

##
6 SN Observation missed deadline, listed as following:

2012cg, 2013ar, 2013be, CSS130218:092354+385837, CSS130317:082848+293031, SNhunt175

0 SN Observation didn't finish completely, listed as following:

N/A

62.50% SN Observations fully scheduled on time!
##

[Plan 3 generated by SOPG, SNeT]

##
2013-03-21: CSS130218:092354+385837, 2013bo, 2013bt, 2013bq, PTF13asv, SNhunt175, 2013ar, 2013bs
2013-03-26: CSS130218:092354+385837, 2013bo, SNhunt175, 2013bt, 2013bq, 2013ar, 2013bs, PTF13asv
2013-04-20: 2013ar, 2013bt, 2013bo, 2013bq, SNhunt175, CSS130218:092354+385837, 2013bs, PTF13asv
2013-04-25: CSS130317:082848+293031, 2013bs, 2013bo, 2013bq, 2013bt, 2013cb, PTF13asv
2013-04-29: 2013bs, 2013cs, 2013cb, 2013bt, CSS130317:082848+293031, 2013be, PTF13ayw
2013-05-19: CSS130317:082848+293031, 2013cs, 2013da, 2013bs, 2013be, 2013cb, PTF13ayw
2013-05-22: 2012cg, 2013bs, 2013ck, 2013be, PTF13ayw, 2013cs
2013-05-23: 2013da
2013-05-28: 2012cg, 2013cs, PTF13ayw, 2013ck
2013-06-13: 2013ck, 2012cg
2013-06-17: 2012cg, 2013ck
##

##

Name	z	B-Peak	2013-03-21	2013-03-26	2013-04-20	2013-04-25	2013-04-29	2013-05-19	2013-05-22	2013-05-23	2013-05-28	2013-06-13	2013-06-17
2012cg	0.001	2012-04-01	-	-	-	-	-	-	41	-	41	41	41
2013ar	0.06	2013-03-23	65	65	65	-	-	-	-	-	-	-	-
2013be	0.065	2013-04-16	-	-	-	-	77	77	77	-	-	-	-
2013bo	0.036	2013-04-11	35	35	35	35	-	-	-	-	-	-	-
2013bq	0.06	2013-04-24	50	50	50	50	-	-	-	-	-	-	-
2013bs	0.027	2013-04-21	38	38	38	38	38	38	38	-	-	-	-
2013bt	0.036	2013-04-26	32	32	32	32	32	-	-	-	-	-	-
2013cb	0.054	2013-05-12	-	-	-	47	47	47	-	-	-	-	-
2013ck	0.049	2013-05-12	-	-	-	-	-	-	58	-	58	58	58

2013cs	0.009	2013-05-24	-	-	-	-	27	27	27	-
27	-	-								
2013da	0.021	2013-06-06	-	-	-	-	-	27	-	27
-	-	-								
CSS130218:092354+385837	0.05	2013-03-10	30	30	30	-	-	-		
-	-	-	-	-						
CSS130317:082848+293031	0.08	2013-03-19	-	-	-	82	82	82		
-	-	-	-	-						
PTF13asv	0.035	2013-05-11	50	50	50	50	-	-	-	-
-	-	-								
PTF13ayw	0.053	2013-05-18	-	-	-	-	53	53	53	-
53	-	-								
SNhunt175	0.0409	2013-03-18	43	43	43	-	-	-	-	-
-	-	-								

##

##
7 SN Observation missed deadline, listed as following:

2012cg, 2013ar, 2013be, 2013ck, CSS130218:092354+385837, CSS130317:082848+293031, SNhunt175

1 SN Observation didn't finish completely, listed as following:

2012cg

56.25% SN Observations fully scheduled on time!
##

BIBLIOGRAPHY

[1] http://en.wikipedia.org/wiki/Supernova.

[2] Composite View of Kepler's Supernova Remnant – SN 1604. http://www.spitzer.caltech.edu/images/1278-ssc2004-15a1-Composite-View-of-Kepler-s-Supernova-Remnant-SN-1604.

[3] Astronomy Today Volume 2: Stars and Galaxies with MasteringAstronomy, Seventh Edition. Chaisson & McMillan, 2011.

[4] Skyalert.org. http://skyalert.org/.

[5] The Astronomer's Telegram. http://www.astronomerstelegram.org/.

[6] IAU Central Bureau for Astronomical Telegrams. http://www.cbat.eps.harvard.edu/index.html.

[7] AstroShelf Project. http://db.cs.pitt.edu/group/projects/astroshelf.

[8] Panayiotis Neophytou, Roxana Gheorghiu, Rebecca Hachey, Timothy Luciani, Di Bao, Alexandros Labrinidis, G. Elisabeta Marai, and Panos K. Chrysanthis. AstroShelf: Understanding the Universe Through Scalable Navigation of a Galaxy of Annotations. Proc. of the 31st ACM International Conference on Management of Data (SIGMOD), pp. 1-4, Scottsdale, Arizona, May 2012.

[9] Google Reader. http://en.wikipedia.org/wiki/Google_Reader.

[10] Qinglan Li, Alexandros Labrinidis, and Panos K. Chrysanthis. ViP: a User-centric View-based Annotation Framework for Scientific Data. The 7th Hellenic Data Management Symposium (HDMS'08), pp. 1-12, Crete, Greece, July 2008.

[11] Qinglan Li, Alexandros Labrinidis, and Panos K. Chrysanthis. ViP: a User-centric View-based Annotation Framework for Scientific Data. Proc. of the 20th International Conference on Scientific and Statistical Database Management (SSDBM'08), pp. 295-312, Hong Kong, China, July 2008, DOI:10.1007/978-3-540-69497-7_20.

[12] Bin Packing Problem. http://en.wikipedia.org/wiki/Bin_packing_problem.

[13] F. Lindh, T. Otnes, and J. Wennerstrom. Scheduling Algorithms for Real-Time Systems. Department of Computer Engineering, Malardalens University, Sweden.

[14] Combinatorial Optimization. http://en.wikipedia.org/wiki/Combinatorial_optimization.

[15] Christos H. Papadimitriou and Kenneth Steiglitz Combinatorial Optimization : Algorithms and Complexity; Dover Pubns; (paperback, Unabridged edition, July 1998) ISBN 0-486-40258-4.

[16] Knapsack Problem. http://en.wikipedia.org/wiki/Knapsack_problem.

[17] M. R. Garey and D. S. Johnson. Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman, New York, 1979.

[18] L. Hall. Computational Complexity. The Johns Hopkins University.

[19] Sahni, Sartaj. "Approximate algorithms for the 0/1 knapsack problem." Journal of the ACM (JACM) 22.1 (1975): 115-124.

[20] Martello, Silvano, David Pisinger, and Paolo Toth. "Dynamic programming and strong bounds for the 0-1 knapsack problem." Management Science 45.3 (1999): 414-424.

[21] Norvig, Peter. Paradigms of Artificial Intelligence: Case Studies in Common LISP. Morgan Kaufmann Publishers, Inc, 1992.

[22] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), Introduction to Algorithms (2nd ed.), MIT Press & McGraw–Hill, ISBN 0-262-03293-7. Especially pp. 323–69.

[23] Stuart J. Russell and Peter Norvig. Artificial Intelligence - A Modern Approach (Third Edition). Especially Chapter 4 - Beyond Classical Search.

[24] Earliest Deadline First. http://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling.

[25] Shortest Remaining Time. http://en.wikipedia.org/wiki/Shortest_remaining_time.

[26] Least Slack Time. http://en.wikipedia.org/wiki/Least_slack_time_scheduling.

[27] Bishop, Christopher M. "Chapter 4. Linear Models for Classification". Pattern Recognition and Machine Learning. Springer Science+Business Media, LLC. pp. 217–218. ISBN 978-0387-31073-2.

[28] Bishop, Christopher M. "Chapter 10. Approximate Inference". Pattern Recognition and Machine Learning. Springer Science+Business Media, LLC. pp. 498–505. ISBN 978-0387-31073-2.