CHAPTER 1

AUTOMATIC FITTING OF OPTICAL TYPE IA SUPERNOVA SPECTRA - THE DALEK PROJECT

The last chapters (Chapters **??** were dedicated to the hunt for donor stars and did not use the measurements from the SN Ia-phenomenon itself. In this chapter we will describe the extraction of yields and energies from optical spectra.

The two main sources of information in spectra, are the spectra themselves as well as their time evolution. There have been a few attempts to extract the details of the stellar explosions from one or two of these sources. All of them employ the technique of fitting the spectra using synthetic spectra. One of the main parts is the radiative transfer program that creates the synthetic spectra. There are several different radiative transfer-codes in the community.

Fisher (2000) wrote a very simple radiative transfer code called SYNOW. SYNOWis a highly parametrized code and thus is mainly used for line identification rather than actual fitting of supernova spectra. It runs The main code (henceforth ML MONTE CARLO) used in this work is an evolved code of Mazzali & Lucy (1993); Mazzali (2000). Compared to the SYNOW-code the ML MONTE CARLO-code calculates a radiative equlibirum tepmerature and uses this to compute internally consistent ionization ratios. In addition ML MONTE CARLOtakes electron scattering into account as well as allowing for photon branching.

Codes such as PHOENIXHauschildt & Baron (1999), SEDONA Kasen et al. (2006) and ARTIS Kromer & Sim (2009) are powerful 3D radiative transfer codes. They are the most "physical" codes available but take hours on supercomputers to produce spectra. These codes, however, are not feasible for fitting observed spectra as they take too long for each iteration.

The main aim of this work is to automatically fit the torrent of observed spectra expected from the current and next generation of supernova searches. We opted to use the ML MONTE CARLO-code as it provides a good compromise between speed and "realism".

In section **??**we will introduce a the inner-workings of the ML MONTE CARLO-code. We will discuss the properties of the search space in **??** and will introduce our optimisation strategies in **??**. Finally we will conclude and give an outlook over future work for this unfinished project in section **??**.

1.1. The ML MONTE CARLO-Code

The supernova can be divided in two different phases: the photospheric phase and the nebular phase. The ML MONTE CARLO-code only models the photospheric phase. In this photospheric phase the supernova is treated like a sharp photosphere emitting a black-body spectrum with a fast moving layer of ejecta above that.

There are many physical processes in radiative transfer. Of those the Bound-free opacity has the biggest contribution to the final spectrum. In addition, Thompson scattering is thought to have an important effect in redistributing the flux. As ML MONTE CARLOis required to run fast only Bound-Free opacity as well as Thompson scattering is implemented in the code.

Unlike stellar atmospheres in supernova ejecta one needs to consider the photon's doppler shift in relation to the surrounding medium. One major assumption that the code makes is that of the Sobolev approximation. This means that at the interaction between photon and line resonance happens only at one specific point (thus disregarding any broadening effects to the line). For example a photon in free flight from the photosphere will be able to interact with resonance lines of lower and lower frequencies. The Sobolev approximation makes the code relatively fast

Another assumption that ML MONTE CARLOmakes is that the ejecta is in homologous expansion. This means that the velocity is a linear function of the radius:

$$v = r/t.$$

Combining both the Sobolev approximation with the assumption of homologous expansion yields this relatively simple formulat for line opacities:

$$\tau_{ul} = \frac{\pi e^2}{m_e c} f \lambda t_{\exp} n_l \left(1 - \frac{g_l n_u}{g_u n_l}\right),$$

where τ_{ul} denotes the opacity going from the u-state to the l-state, e is the electron charge, m_e is the electron mass, f is the oscilator strength of the line, λ denotes the wavelength, $t_e xp$ the time since explosion, n_x the number of atoms in the state x and g_x is the statistical weight of the state x. Both homologous expansion and Sobolev approximation have their caveats. In the case of homologous expansion it is thought to be a very good approximations after the first few minutes after the explosion. The main caveat for Sobolev approximation is that a line is not a delta-function, as assumed in the Sobolev approximation. If too strong bound-bound lines are close in frequency space it can lead to the first line shielding the second line. In summary for fast supernova fitting both approximations seem to still allow for a relatively well fitting spectrum.

We have discussed the propagation of the photons in the plasma but have not discussed the state of the plasma yet. The simplest assumption for the state one can make is local thermodynamic equilibrium. In this case the Boltzmann formula describes the level populations in a single ion:

$$\frac{n_j}{n_{\text{ground}}} = \frac{g_j}{g_{\text{ground}}} e^{-(\epsilon_j - \epsilon_{\text{ground}})/kT}$$

Similarly we can calculate the ionization state using the Saha-equation:

$$\frac{N_j}{N_{j+1}} = n_e \frac{U_j(T)}{U_{j+1}(T)} C_I T^{-3/2} e^{\chi/kT},$$

where U_j is the partition function and C_I is a constant. As the ionization likelyhood depends on the internal electronic state of the atom the partition function sums up over the different states:

$$U_j = \sum_i g_{i,j} e^{-\frac{E_{i,j}}{kT}},$$

where i describes the excitation states and j the ionization states. The other symbols have their usual meaning. The sum normally diverges slowly so one in practice just sums up until a highly excited state.

The ML MONTE CARLO uses the so called *nebular approximation* which will calculate the excitation and ionization state of the SNe at nearly LTE cost. In this nebular approximation they introduce a dilution factor W. This is a purely geometrical factor. Treating the photosphere as a point source the factor would result in $W = 1/r^2$ with r being the distance from the center. As the photosphere is expanded the dilution factor has a slightly more complex formula. An important point to note is that purely theoretical at the photosphere the dilution factor is 0.5.

The mean intensity for the supernova at a specific zone is given as:

$$J = WB(T_R),$$

where T_R is the radiative temperature. The radiative temperature is estimated in the ML MONTE CARLO by matching the mean frequency of $B(T_R)$ with the mean frequency of the photon packets in the current zone (Wien approximation). W is chosen so that the frequency-integrated intensity matches the photon distribution.

Using W and J one now can calculate the electronic and ionization states of the plasma:

$$\frac{n_j}{n_{\text{ground}}} = W \left(\frac{n_j}{n_{\text{ground}}} \right)_{T_R}^{\text{LTE}}$$

and

$$\frac{N_j}{N_{j+1} n_e} = W \left(\frac{N_j}{N_{j+1} n_e} \right)_{T_R}^{\text{LTE}}$$

In the simplest case we can treat the ejecta as homogeneous in temperature and abundance. For now we will also assume a pure scattering line interaction. This means that the photon is absorbed at a resonance frequency and then instantaneously reemitted with the same frequency into a random direction. This in in contrast to photon branching which we will discuss later.

We assume a time since explosion t_e, a photospheric velocity v_{ph}, $L_{\text{bol}}$and an abundance distribution for the chosen elements. W7 (Nomoto et al., 1984) is used in the ML MONTE CARLO as a density structure.

The one dimensional model is divided into multiple zones that each have the same abundance but a different density. Using an initial guess of T_{eff} for the photosphere, one can calculate the plasma condition in each shell.

The Monte-Carlo simulation begins. A photon packet is emitted with a random frequency and a random angle drawn from a Blackbody distribution $B(T)$. Each photon packet contains the same energy (more photons per packet in the red than in the blue). An event optical depth is calculated from a uniform random distribution so that $\tau_{\text{event}} = -ln(z); z \in (0, 1]$. In the next step there are three possible outcomes. We calculate the length

of the path (s_e) that the packet can travel freely before τ_{event} is equal to the Thompson scattering opacity $\tau_{\text{event}} = \sigma_T n_e s_e$. Next we calculate the same path length for the lines s_l using as a target opacity $\tau_e + \tau_{\text{line}}$. If both paths are longer than the path to exit the current zone, then the photon exits the current zone and a new Monte Carlo process begins. If however s_e is the shortest then Thompson scattering occurs and the photon is assigned a new direction and a new τ_{event} is drawn and the process begins anew.

In the case of line scattering the excited atom can de-excite through many lines. ML MONTE CARLO randomly chooses a downward transition for the whole packet (taking the appropriate weights into account). The number of photons in the packet is adjusted to ensure that the energy is conserved in the comoving frame.

There are two possibilities for the final fate of the photon. Either it is reabsorbed into the photosphere or is emitted from the supernova.

When initializing the state of the plasma one assumes an initial guess for the photon temperature. The Monte-Carlo simulation is run and records each packet status at the mid-point of each shell. This information is used to calculate a new photospheric temperature and an updated plasma condition (level population and ionization). This procedure is repeated until the photospheric temperature converges. Once convergence is reached the actual Monte-Carlo simulation begins.

The final spectrum is not calculated using the escaping packets. Instead we calculate the optical depths using and then calculate the emerging spectrum using the formal integral. This has the advantage of reducing noise in the spectrum due to Monte-Carlo noise and gives very good results.

A more detailed description of the code can be found in Mazzali & Lucy (1993); Mazzali (2000).

1.2. Manually fitting a Type Ia supernova

When fitting manually there are several features that help guide the direction of the fit. We will attempt to explain by using a spectrum of SN2002bo (cite?????) 10 days before maximum. In this section we will only talk about fits with no abundance stratification. Stephan Hachinger has kindly provided his manually obtained best fitting parameters (for the supernova at this stage (see Figure 1.1). Directly measurable are the redshift of the supernova (and implied distance) and the time of the spectrum relative to maximum. We assume calculate the time since explosion assuming a rise time of 19.5 days. The other parameters are initialized using empirical data.

The chosen fundamental parameters are $\log L/L_\odot = xx$, $v_{\text{ph}} = xxx$. We have listed the non-zero abundances in Table **??**.

The P-Cygni profiles of many features are easily visible. The Calcium line in the blue can be seen to be to blueshifted in relation to the model. This property is not unusual and is thought to come from high velocity component at the outer edge of the ejecta. The next major known discrepancy that can be seen is the excess of flux redwards of ≈ 6200Å. This is a region that usually does not fit well as the underlying black body spectrum overestimates the flux in this region. When fitting manually often one tries to fit the depth the lines instead of the continuum.

There are three main parameters that have the most influence on the overall fit: Luminosity, photospheric velocity and abundance in iron group elements.

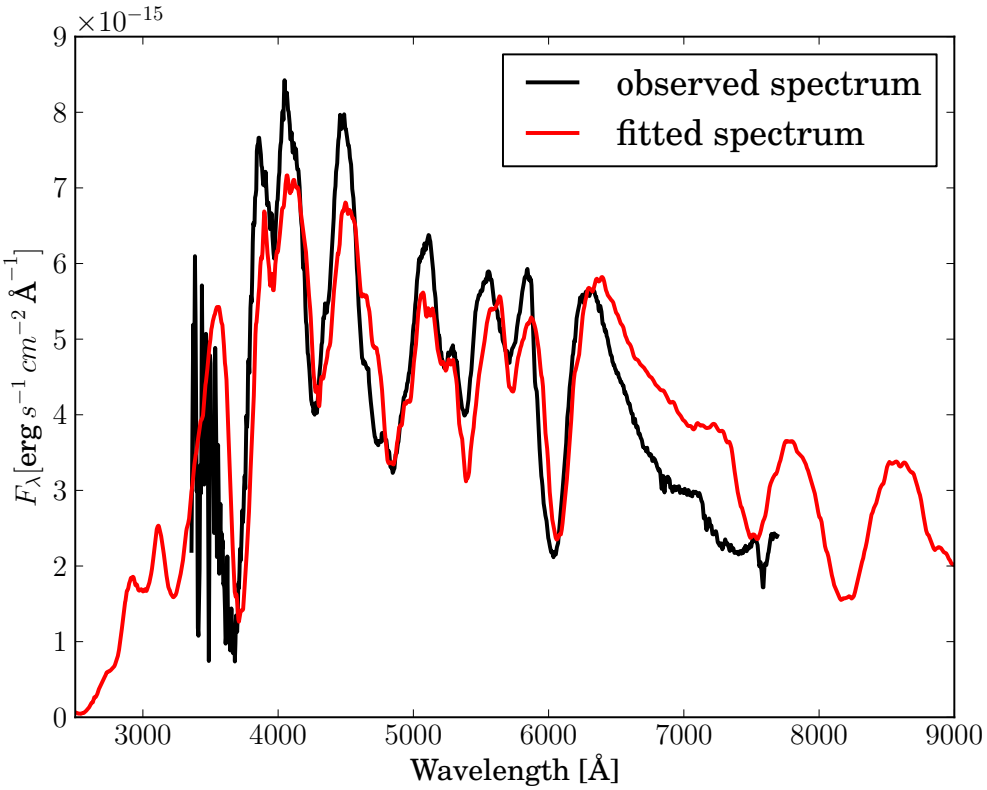

Figure 1.1 example caption

A large offset in L to the best fit paramater is easily visible as a large offset of the continuum (see Figure **??**. Thus it is easy to constrain the parameter space in L initially. L also has influence on the temperature of the model through:

$$L_{\text{bol}} = 4\pi\sigma\, R^2\, T^4 = 4\pi\sigma\, v_{\text{ph}}\, t_{\text{exp}}.$$

Velocity in astronomy is often measured using the doppler shift. In this case however it is hard to measure the photospheric velocity as lines are created at different depths and thus at different velocities. This smears out the line profiles which makes fitting velocities nearly impossible using this technique. The main impact of photospheric velocity is establishing the temperature given a luminosity. A model with a too high photospheric velocity will have expanded more than the real spectrum at that time and will be cooler. This results in a spectrum that is too luminous in the red and not luminous enough in the blue (see Figure 1.2). A secondary effect is that the ion population will be different to the actual supernova and the line spectrum differs. The initial abundances for each fit

Figure 1.2 example caption

are determined by integrating the abundances above the photosphere in the W7 model (Nomoto et al., 1984). The iron group element have a similar influence on the overall flux distribution as the photospheric velocity. As we assume no stable Cobalt and the input parameters for Nickel, Cobalt and Iron are $^{56}\text{Ni}_0$ and $^{56}\text{Fe}_0$ and calculate the abundances

using radioactive decay. Ti and Cr have no easily identifiable single lines in the observed spectra, but provide line blanketing in the blue. We often lock their ratios and only use one abundance as an input parameter.

These elements cause photons to be absorbed in the UV and be reemited in the red. A too high abundance will suppress the flux in the blue too much and will cause the spectrum to be over-luminous in the red (see Figure **??**). Although physically different from the photospheric velocity, phenomenologically these are similar. The degeneracy is broken by identifiable Fe-Lines in the red part of the spectrum as well as the ionization balance determined by the temperature (influenced by photospheric velocity). This near degeneracy causes a very complex search space.

There are six other abundances that are taken into account when fitting: Carbon, Oxygen, Magnesium, Silicon, Sulfur and Calcium. Among these Oxygen plays a special role. It does not have lines except the Oxygen triplet at 7778 Å. In our fitting routine it acts as a buffer element and is assigned the remaining fraction that is left after all elements have been given abundances.

The first element that is usually adjusted from its initial value is Calcium. The Calcium line is relatively easy to identify. Calcium also does not depend an awful lot on the right choice of temperature. One caveat however is that Calcium saturates at a certain point so if the observed line is close to that point one can only extract a lower limit for Calcium.

The next to be adjusted are Silicon and then Sulfur. Both of these elements are linked through nuclear synthesis and we don't expect there to be more Sulfur than Silicon. We also expect no less Sulfur than a third of Silicon. Silicon also provides an important measure for temperature through the ionization balance of doubly ionized Silicon to triple ionized Silicon (see Figure **??**).

Magnesium has one strong feature near 4300 Å, which constrains the Magnesium abundance.

Doubly ionized Carbon has a line at $\approx$ 6300 Å which is often not present or very weak.

Fitting the spectra involves first adjusting luminosity and then IGEs as well as photospheric velocity. This is followed by adjusting the other elemental abundances from the initial W7 values (Nomoto et al., 1984). After the elemental abundances are adjusted we readjust the luminosity, photospheric velocity and IGEagain. This loop is continued until convergence is reached.

One important factor to check is the value of the dilution factor w at the photosphere. Theoretically we would expect this to be close to 0.5 if the fit is sensible.

After having fit one spectrum we can continue to fit the other photpsheric spectra of the same supernova. We expect most parameters to increase or decrease monotonically (luminosity being the obvious exception).

In summary, the fitting of a supernova is a complex procedure and requires a lot of practice. Our initial tests were using very simple methods like Newton-Raphson and other gradient methods. We quickly discovered the search space is too complex and evaluation time takes too long (each spectrum roughly one minute on a modern computer) to use these simple methods.

1.3. Genetic Algorithms

Finding optimal solutions in complex search spaces is one of the main fields in numerical mathematics. They have wide ranging applications in engineering, bio sciences and physical sciences. There have been several advances in the last decads to optimization. Among them is the remarkable feat of "solving" the long-standing problem of the traveling salesman with simulated annealing (Kirkpatrick et al., 1983).

Holland (1962) Another major accomplishment was the development of evolutionary algorithms and subsequently genetic algorithms. The idea of an algorithm which imitates the principal of natural evolution was first introduced by ??? give story of rechenberg optimizing wind tunnel ??? Ingo Rechenberg (Rechenberg, 1973). These evolutionary algorithms have since become a sizeable subfield of numerical optimization. Genetic algorithms a subclass of evolutionary algorithms were then further developed by John Holland and David Goldberg (Goldberg, 1989).

When conquering the search space optimization algorithms have two (sometimes conflicting) goals: exploiting good leads while still exploring the search space sufficiently. Simple algorithms like Hillclimbing (randomly selecting a point in the neighbourhood and switch if it is "better" than the current one) will exploit good leads but will neglect to explore the search space. This often leads to be stuck at extrema. Whereas random searches are excellent at exploring the search space but will fail to quickly converge on an optimal solution. Genetic algorithms do strike the balance between exploration and exploiting the current best solution. """" maybe include: Dependent variables present special problems for optimization algorithms because varying one variable also changes the value of the other variable. For example, size and weight of the car are dependent. Increasing the size of the car will most likely increase the weight as well (unless some other factor, such as type of material, is also changed). Independent variables, like Fourier series coefficients, do not interact with each other. If 10 coefficients are not enough to represent a function, then more can be added without having to recalculate the original 10. In the GA literature, variable interaction is called epistasis (a biological term for gene interaction). When there is little to no epistasis, minimum seeking algorithms perform best. GAs shine when the epistasis is medium to high, and pure random search algorithms are champions when epistasis is very high (Figure 2.5). cite haupt & haupt (book on harddrive) """""""""" Genetic algorithms are a heuristic search technique to find optimal solutions in n-dimensional search spaces. We will consider a function $f(\vec{x})$ with the multi-dimensional solution $\vec{y}$. In terms of genetic algorithms we call $\vec{x}$ genotypes and the solution vector $\vec{y}$ phenotype (similar to biology where the $\vec{x}$ describes the DNA sequence but the solution is can not be thought of as a vector). In addition we define a fitness function $g(\vec{y}) = s$ where s is a scalar. It is our goal to choose the input parameters $\vec{x}$ to maximize s. Following the notation of (Michalewicz, 1994) we introduce the population $P(t)$ with the individuals (sometimes referred to as genomes) $\{p_1^t, \ldots, p_n^t\}$, where t denotes the iteration. Each individual (p_i^t) is a data structure consisting of a vector $\vec{x}_i$ and its corresponding fitness scalar s_i. When we speak of evaluating p_i^t we mean that we use $g(f(\vec{x}_i)) = s_i$ to determine the fitness. A new population $P(t+1)$ is formed by choosing, in the *select step*, the more fit individuals. Some or all of the new population undergo transformations (recombine step). These transformations are called genetic operators. There are unary transformations, which create new individuals by small changes in single individuals called mutations. Higher

order transformations called crossovers combine the traits of multiple individuals to form a next generation individual. After the new population has been created in the recombine step, it is evaluated and the *select step* begins anew.

This procedure is repeated until the best individual has reached a certain threshold fitness (see Algorithm **??**).

Algorithm 1 Structure of a genetic algorithm

```
t ← 0
initialize P(t)
evaluate P(t)
while (not termination condition) do
  t ← t + 1
  select P(t) from P(t − 1)
  recombine P(t)
  evaluate P(t)
end while
```

To be solvable by a GAthe problem needs to meet the following requirements:

- a genetic representation of the search space (e.g. a vector)
- a function that can calculate a fitness for a genetic representation
- transformations that create a new population out of selected members of the old population
- a method of creating an initial population

If these are fulfilled one can start constructing a GAfor the chosen problem. When constructing a new GAfor a problem there are multiple steps. First one needs to find a suitable genetic representation for each solution in the search space.

Genetic Representation: There are two main ways to represent a genome: binary-encoding and value encoding (sometimes called gray encoding). Binary encoding was the form of encoding used in early genetic algorithms. It offers significant advantages when trying to optimize very simple problems.

In one-dimensional problems, for example, value encoding would only offer one gene, whereas binary encoding offers a number of genes depending on the requested precision of the value. There are however many problems with binary encoding. The so called *hamming cliff* describes the problem that a simple bit-flip at one high encoding bit (occurring in some GA operations) can dramatically change the encoded value Chakraborty & Janikow (e.g. 2003). This can improve covering of search space but also can hinder the code from converging quickly. When using binary encoding for many input variables the genomes can get incredibly large. GAs have been shown to perform poorly for very large genomes. quicker convergence michalewicz page 82 chpter 5.5

Value encoding often is a natural way to encode the parameters of a problem. In contrast to binary encoding the genetic operators are often much more problem specific. It seems that for the moment value encoding is the preferred method in many cases (e.g. Janikow & Michalewicz, 1991; Wright, 1991).

Finally, an important encoding to mention (which does not apply to our example problem) is that of permutation encoding. In the famous case of the travelling salesman problem (henceforth TSP) one tries to find the shortest route between n cities. In this case each city can only be visited once and the route must end in the starting city. There are many algorithms that can solve this problem. Brute force attempts scale with $O(n!)$ which make them unfeasible. A genetic algorithm can solve this problem by encoding the order (permutation encoding) in which the cities are visited in each genome. There are special genetic operators for permutation encoding. For the TSP there are better algorithms like dynamic programming which can solve the problem with a complexity of $O(n^2 2^n)$ (see Figure 1.3).

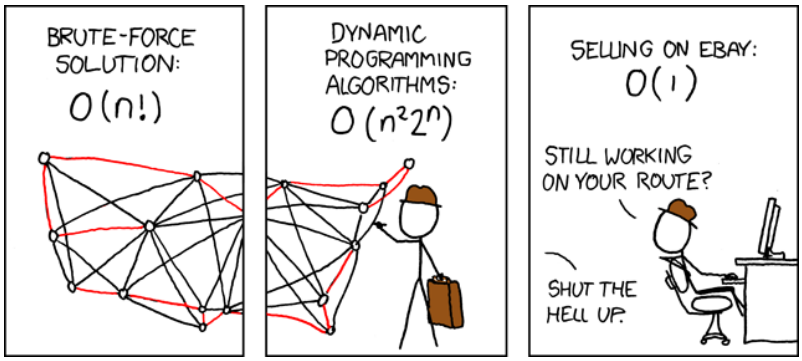

Figure 1.3 The death of the TSP with the advent of online sales. (reproduced with kind permission by xkcd.com)

In the case of a simple vectorized function or the TSP once can now directly calculate the fitness value. In the case of our astrophysics example we need to first transform the genome (genotype) to a synthetic spectrum (phenotype) before being able to calculate the fitness.

Fitness Function Calculating the fitness of an individual is very dependent on the problem under consideration. In the case of our spectrum fitting problem we can simply calculate the root-mean-square for each individual. Finding a good fitness function in any optimization algorithm is one of the most complex tasks. It might not always be possible to find one fitness value that describes the problem but some problems might need many. This is generally referred to as multi-objective optimization and is an active field in GA-research (Konak et al., 2006). Fitness scaling applies a simple transformation to the fitness of all members of the population (e.g. a linear transform). Many selection processes are not able to accept negative and positive values for fitness so fitness scaling alleviates such a problem. In addition, it happens sometimes in early generation that a few individuals already have very high fitness. These "superindividuals" then dominate the selection process and subsequently reduce the genetic breadth drastically. This can in later generations significantly hamper the progress of finding an optimal solution (such a problem was encountered in this work and is described in Section 1.4).

In summary, fitness functions and scaling are a very crucial part of a successful algorithm. For a description of different scaling methods please refer to **?** ??? select some introduction referebces from this paper???.

Initial Population The most basic method of selecting the initial population is to draw individuals uniformly and randomly from the entire search space. It depends then on the population size in relation to the search space size how well this search space is sampled. For a small population size and a large search space the GA could very well find a local optimum rather than the global one. We have in our work chosen a population time which is roughly 15 times bigger than the number of genes. Choosing the initial population using prior knowledge will also improve the performance of the GA. Rather than drawing uniformly random we would draw randomly from a probability distribution covering the search space.

Selection process There are many different approaches for selecting individuals from the current population to create the next population. Before selecting individuals we can make coarse selection on the entire population. One selection that is often performed is that of *elitism* in which a fraction of the fittest individuals is selected to advance to the next generation without being altered. On the other hand one can discard a fraction of the least fit individuals. These discarded members won't be used in the recombination step. The remaining population is called the mating population

After we have performed a coarse selection on the population we start with the recombination step. The first action in the recombination step is the selection of two or more individuals from the mating population and add them to a mating pool. In all our next examples we will assume a mating-pool with only two slots (similar to two parents in biological reproduction). Once the mating pool is filled a new individual is created by combining the genetic information of all members of the mating pool.

There is a multitude of options for selecting members from the mating population and adding them to a mating pool (**?**, for an overview see). We will describe only a few of them. The most widely used of the selection algorithms is *roulette wheel selection* (see Figure **??**). *Roulette wheel selection* is in the class of *fitness proportional selection*. In addition to *fitness proportional selection* there is *rank selection*. In rank selection the fitness only influences the rank of the individuals (see Figure **??**). The previous described fitness scaling and rank selection have similar effects. In *tournament selection* we randomly select two individuals and compare those. The fitter of those two individuals is selected and added to the mating pool.

There are multiple steps for creating a new population from the current mating population. First we select two individuals (or more) using our selection process (e.g. *roulette wheel selection*) and place them in a mating pool. The reader should notice that the same individual can be in the mating pool twice! We create one or multiple new individuals (children) from this mating pool and place them in the new population. The current mating pool is disbanded and a new one is formed. These steps are repeated until the new population has the same number as the old population (minus the number of individuals that automatically advanced to the new population through *elitism*). There are two main processes to create a new individual from a mating pool: crossover and mutation.

Crossover and mutation We assume a mating pool with two individuals. The simplest form of a crossover is the single-point crossover (see Figure **??**). A random integer $r \in [1, N-1]$, where N describes the number of genes per genome is selected. The new individual is created out of the first r genes from the first parent and the last $N-r$ genes

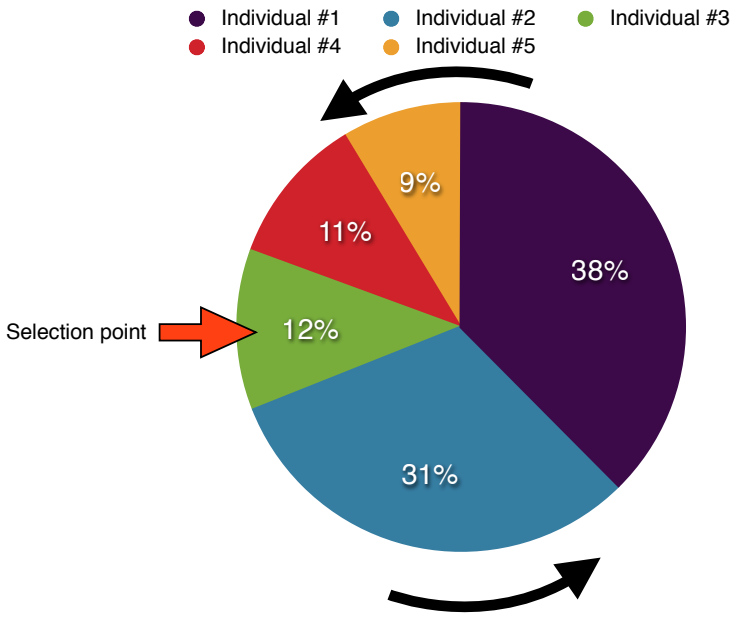

Figure 1.4 example caption

from the second parent. Now it is trivial to create a second child (using the same random number r) from the mating pool by just switching first and second parent. Two-point crossover is very similar to single-point crossover. Two random numbers are selected and the crossover occurs at these places (see Figure **??**). (N-1)-point crossover is also called uniform crossover. In that case each gene is selected randomly with equal chance from the parents. Arithmetic crossovers use a function to calculate the new gene from each of the parent genes. For a value encoding this function could be the mean. For a binary encoding the function could be the *and* operator. ???should I include example??? One can mix arithmetic and binary crossovers. For example we select the r genes from the first parent and create the last $N - r$ genes by taking the mean of first and second parents genes. After the new child or children have been created through crossovers it is subjected to the mutation operator. There is a chance for each individual gene (often chosen to be less than 5%) that it is altered. For bit encoding this altering is a simple inversions. There are many options for value encoding. For example one can add or multiply with a random number. Once this step is complete the children are added to the new population.

1.3.1. A simple example

We will illustrate the use of a GA on a simple astrophysical problem. The task at hand is to fit an observed spectrum with a synthetic spectrum. The input parameters for this synthetic spectrum are$T_{\rm eff}$, $\log g$, [Fe/H], α-enhancement, $V_{\rm rad}$and $V_{\rm rot}$. The simplest genetic representation of this is simply a vector $\vec{x} = (T_{\rm eff}, \log g, [Fe/H], \alpha, V_{\rm rad}, V_{\rm rot})$. $\vec{x}$ is the *genotype* of the individual. The resulting synthetic spectrum is called the *phenotype*.

For this relatively small number of genes a population size of 75 should suffice. The first step is drawing an initial population. We will draw uniformly randomly from the search space: $T_{\rm eff} \in [2000, 9000]$, $\log g \in [0, 5]$, $[Fe/H] \in [-5, 1]$, $\alpha \in [0, 0.4]$, $V_{\rm rad} \in [-100, 100]$

and $V_{\rm rot} \in [0, 200]$. We compute the synthetic spectrum for each individual. The fitness of each individual is the inverse of the root-mean-square of the residuals between the observed and the synthetic spectrum. In the select step we will first advance 10 % of the fittest individuals to the next population unaltered (*elitism*). For the next population to be complete we need $75 - 8 = 67$ individuals which are created through mating. We select 2 individuals through *roulette wheel selection* and place them in the mating pool. A single crossover point is randomly selected and the child is created. Before being placed in the new population the mutation operator is applied but has a very small chance to mutate any of the child's genes (in this case we choose 2%). This mating step (see Figure **??**) is repeated 67 times. The new population now consists of the 8 fittest individuals of the old population and 67 new individuals created by mating. We will then start again to compute the synthetic spectrum and the resulting fitness for each individual of the new population. This loop is continued until one individual or a whole population has reached a preset convergence criterium.

1.3.2. Convergence in Genetic Algorithms

A key problem with all optimization algorithms is the premature convergence on a local optimum. The more complex the search space and the more interlinked the parameters are the more likely it is that traditional search routines will fail. GAs are inherently better at bypassing local optima but are in no way immune to this problem. Premature convergence, as finding a local optimum is called in GA-terms, is a well known problem A problem that is intrinsic to the GAs is that for in traditional implementation they will never fully converge. The algorithm will get close to the optimum but due to continued mutation of the individuals the GA will in most cases not reach it. To alleviate this problem some authors suggest switching to a different algorithm when close to the optimum solution, whereas others suggest changing the mutation rate over time (see Rudolph, 1994, and references therein). A definite advantage of GAs is their parallel nature. This makes it easy to explore large search spaces in the era of increasingly parallel computing.

Although there are many advantages to using GAs, one of the unsolved problems is determining a mathematical description for their convergence. The predominate schemata theory explains only a subset of the intrinsic complexity of GAs.

1.3.3. GA theory schemata theory

The schemata-theory first described by Holland (1975) is one of the accepted theoretical interpretations of GAs. There is some criticism and it is known that this theory only explains part of the complexity that are inherent to GAs (see Whitley, 1994, and references therein). We will describe the basic concepts of schemata using an example in binary encoding (notation adapted from Goldberg, 1989). A schema or similarity template is formed by adding an extra letter to the binary alphabet denoted by $*$. Using the ternary alphabet $0, 1, *$ we can now describe pattern matching schemata were the $*$ letter can be thought of as *don't care*-symbol. The schemata $0, 1, 0, *$ matches both the string $0, 1, 0, 0$ and $0, 1, 0, 1$. The order of a schema is defined as how many places are not filled by the $*$-symbol. Schematas provide a powerful way to describe similarities in a set of genomes. So a whole population samples different a range of different schematas . Essentially low-order and above-average schemata, called building blocks, receive exponentially increasing trials in

subsequent generations of a genetic algorithm. Michalewicz (1994) describe the workings of a GA in the following way: "A genetic algorithm seeks near=optimal performance through the juxtaposition of low-order, high-performance schemata, called building blocks". This schemata-theory is the standard explanation for GAs, there are however some examples that violate this (see chapter 3 of Michalewicz, 1994, for some examples).

1.4. Genetic Algorithms to fit Type Ia Supernovae: The Dalek Code

1.5. Introduction

As described in Section 1.2 the search-space for fitting SNe Ia is very complex and vast. The parameters that need to be fit are luminosity, photospheric velocity ($v_{\rm ph}$), Carbon, Magnesium ,Silicon, Sulfur, Calcium, Chromium, initial Ni (Ni_0) and primordial iron (Fe_0). The day since explosion as well as the luminosity distance (for scaling purposes) are given. In addition, the evaluation for each spectrum takes roughly one minute on a modern computer. This makes it hard to quickly try out different optimization strategies. We initially tried a Newton-Raphson approach with multiple phases. In the first phase the algorithm would adjust luminosity and normally came close to the optimum. In a second phase we tried to let the algorithm re-adjust luminosity, then photospheric velocity and last IGEs. This was modelled after the manual approach that is often taken. It took relatively long and would not really converge. The search space for such methods is far to complicated. In addition we were limited to one processor as the code is not parallel. GAs seemed the perfect choice. They are easy to implement, can be easily parallelised and are relatively prone to local minima. For most of our optimization tests we worked with the supernova SN 2002bo (**?**). This supernova is believed to be a "Branch"-normal and is spectroscopically well sampled. For most of our tests we worked on the $t_{\rm exp}$=10.4 days spectrum, but did not make any other special assumptions about the spectrum. We have used some other spectra and will mention this where applicable.

1.6. The Dalek Code

After the failed attempts of the multi-phase Newton-Raphson fitter we launched straight into GAs. As described in Section **??** one needs to first create an initial population. One can easily draw uniform randomly for luminosity and $v_{\rm ph}$. This method does not work for the elemental abundances as the sum of the abundances needs to add up to 1. In addition, for many normal supernovae the W7-abundances (Nomoto et al., 1984) give a good starting point. A population that is distributed around the optimum values does converge quicker than a uniformly random one. To calculate these abundances we need to know the photospheric velocity. Using data from **?** we have estimated an empirical relationship between the time since explosion and the photospheric velocity (see Figure **??**). This will serve as a rough first guess.

Once we know a $v_{\rm ph}$-estimate we can determine at what depth the photosphere is located in the W7-model. We use this point and integrate outwards to find our initial abundances.

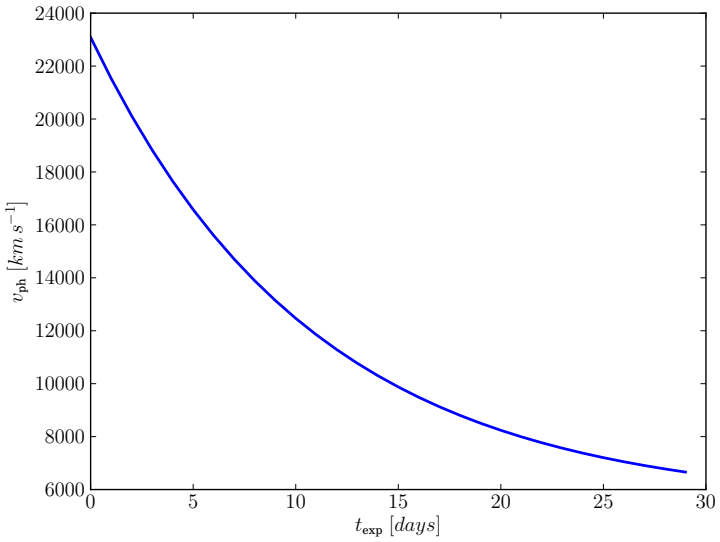

Figure 1.5 example caption

When creating our initial population we draw uniformly random for a preselected luminosity range (for the moment determined manually), a v_{ph}-range 4000 km s^{-1}above and below the v_{ph}-estimate and asymmetric log-normal centered on the abundance estimate from W7. Creating the abundance values brings some problems with it. There are several bounds when it comes to the initial abundance ratios. We have a minimum bound for each abundance (established as the smallest abundance an element can have by ML MONTE CARLO) at 10^{-7} and a maximum bound which is determined by the requirements that all abundances need to add up to 1. If we would draw the abundances every time in the same order it would mean that the last abundances to be drawn have a very low chance of obtaining a high value. Thus we randomize the order of drawing the abundances. Before we let an individual into the initial population we also check some abundance ratios by employing generous limits that we think are allowed by explosive nucleosynthesis of a CO-WD.

- C < 12.5 %
- Si > 1 %
- Ca < 5 %
- Ti + Cr < 1 %
- Ni_0 < 80 %
- C < O
- 1 < Si/S-ratio < 10
- Fe_0/Ni_0-ratio < 10
- Cr/Ni_0-ratio < 10

If the newly created individual does not conform with these rules it is discarded and a new one drawn. This process is repeated until we reach our population size which we chose to be 150.

Once the initial population is created it is distributed among the compute nodes by the Dalek Code. As we need to explore a large parameter space we use a distributed computing approach. The scheduler part of Dalek Code distributes parameter sets to spectrum synthesizers working on different machines across the network. The difficulty is (and the reason that we didn't use a standard implementation like MPI) that we need the Dalek Code to reach processors on many different architectures and operating systems. This enables us to harness a large fraction of the institutes computational power. For the order of execution we have chosen a very simple scheduling algorithm that maintains a FiFo batch-queue for the parameter sets and distributes them among the pool of workers.

The resulting spectra are then subjected to a fitness function. The determination of the fitness function is one of the most crucial choices for the GA to work. The correct fitness function is still a field of active research in the Dalek Code. As an initial approach we calculated the mean-square of the residuals remaining from subtracting the observed spectrum and the spectrum of the current individual. This approach has two main issues: For almost all observed spectra in the early phase the fitted spectrum has a large continuum excess beyond 6500 Å(see Figure . As described previously this is a known issue of the ML MONTE CARLO. This large difference in the infrared means that the Dalek Code will try to optimize this large offset and pay less regard to a good fit in the rest of the spectrum. To alleviate this we have

Initial population. show vph graph. select from w7 integrate outwards. create children randomly; complex needs to work out and obey rules otherwise dead: describe process look at python code. ....

reference back to parallelity and describe launching to multiple (non homogenious computers) simple fifo.

fitness function describe chi**2 and then describe w-parameter and ir excess mention figure.

selection process rws; crossover uniform, mutation non-uniform different mutations for different parameters. child survival rate small at beginning.

describe principal evolution; show 2002bo different channels; population plot histtogram. Fitting works sometime find good solution. discuss time and computer demand. discuss what has been tried

problems?

suggest that on the right way, now working with experts of genetic algorithms. describe differential evolution.

ask james about machine learning techniques....

1.7. Conclusion

GA powerful tools, initially very good but require fine tuning for each problem.

A conventional, but not entirely accurate, interpretation of the NFL results is that "a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration" http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

Continue on investigation with experts on GA. try some different approaches. Fast exploration of search space through grid and GA.

BIBLIOGRAPHY

Chakraborty, U. K. & Janikow, C. Z. 2003, Information Sciences, 156, 253 , evolutionary Computation [LINK]

Fisher, A. K. 2000, PhD thesis, THE UNIVERSITY OF OKLAHOMA

Goldberg, D. E. 1989, Genetic Algorithms in Search, Optimization, and Machine Learning, 1st edn. (Addison-Wesley Professional) [LINK]

Hauschildt, P. H. & Baron, E. 1999, Journal of Computational and Applied Mathematics, 109, 41

Holland, J. H. 1962, J. ACM, 9, 297 [LINK]

—. 1975, Adaptation in Natural and Artificial Systems (Ann Arbor, MI, USA: University of Michigan Press)

Janikow, C. Z. & Michalewicz, Z. in , Proc. of the 4th International Conference on Genetic Algorithms, ed. R. K. BelewL. B. Booker (Morgan Kaufmann), 151–157

Kasen, D., Thomas, R. C., & Nugent, P. 2006, ApJ, 651, 366

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983, Science, 220, 671 [LINK]

Konak, A., Coit, D. W., & Smith, A. E. 2006, Reliability Engineering & System Safety, 91, 992 [LINK]

Kromer, M. & Sim, S. A. 2009, MNRAS, 398, 1809

Mazzali, P. A. 2000, A&A, 363, 705

Mazzali, P. A. & Lucy, L. B. 1993, A&A, 279, 447

Michalewicz, Z. 1994, Genetic algorithms + data structures = evolution programs (2nd, extended ed.) (New York, NY, USA: Springer-Verlag New York, Inc.)

Nomoto, K., Thielemann, F.-K., & Yokoi, K. 1984, ApJ, 286, 644

Rechenberg, I. 1973, Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Problemata No. 15 (Stuttgart-Bad Cannstatt: Frommann-Holzboog)

Rudolph, G. 1994, Neural Networks, IEEE Transactions on, 5, 96 [LINK]

Whitley, D. 1994, Statistics and Computing, 4, 65

Wright, A. H. 1991, in Foundations of Genetic Algorithms (Morgan Kaufmann), 205–218