CHAPTER 1

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

The last chapters (Chapters **??**) were dedicated to the hunt for donor stars and did not use the measurements from the SNe Iathemselves. In this chapter we will describe the extraction of yields and energies from optical spectra as well as the automation of this process.

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. SYNOW is a highly parametrized code and thus is mainly used for line identification rather than actual fitting of SN Iaspectra. The main code used in this work is a further development of the code described in Mazzali & Lucy (1993) and Mazzali (2000), henceforth named the ML MONTE CARLO. Compared to the SYNOW-code the ML MONTE CARLO-code calculates a radiative equilibrium temperature and uses this to compute internally consistent ionization ratios. In addition ML MONTE CARLO takes electron scattering into account as well as allowing for photon branching.

Codes such as PHOENIX Hauschildt & 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 to days on supercomputers to produce spectra. These codes, however, are not feasible for fitting many 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 physical "realism".

In section 1.1 we will introduce a the inner-workings of the ML MONTE CARLO-code.

We will discuss the properties of the parameter space on one example fit in **??**. Section **??** is meant as an introduction to genetic algorithms (henceforth GA), which are the optimization strategies of choice in the automatization of the ML MONTE CARLO. We will present the current implementation of the autofitting code in Section 1.4. We have named the code Dalek for no particular reason. Finally we will conclude and give an outlook over future work of this unfinished project in section 1.5.

1.1. The ML MONTE CARLO-Code

As described in Section **??** the supernova can be divided in two different phases: the photospheric phase and the nebular phase. The ML MONTE CARLO-code only creates synthetic spectra for SN Ia in the photospheric phase. In this photospheric phase the supernova is treated like a sharp photosphere emitting a black-body spectrum with a fast moving ejecta-layer on top.

1.1.1. Radiative Transfer

Radiative transfer calculations can be described as the aim to find the wavelength dependent attenuation of a given input Flux F_0. This wavelength dependant attenuation factor is called the opacity τ:

$$F(\lambda) = F_0(\lambda)\, e^{-\tau(\lambda)},$$

where F is the observed Flux and F_0 is an assumed distribution of input flux before being reprocessed by a plasma which imposes the attenuation factor $e^{-\tau}$. There are many physical processes in supernova 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 flux. As ML MONTE CARLO is 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 the interaction between photon and line resonance happens only at one specific point (thus disregarding any broadening effects of 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. This Sobolev approximation is one important factor of making the code fast yet still reproducing the observed spectra rather well.

In addition the ML MONTE CARLO assumes the ejecta to be 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 formula for line opacities:

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

where τ_{lu} denotes the opacity going from the lower state to the upper state of an atom, e is the electron charge, m_e is the electron mass, f is the oscillator strength of the line, λ

denotes the wavelength, t_{exp} 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.

Although producing relatively good synthetic spectra, the assumptions of homologous expansion and Sobolev approximation have their caveats. In the case of homologous expansion it is thought to be a very good approximations in 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 very 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 (henceforth LTE). In this case the Boltzmann formula describes the level populations in a single ion:

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

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 N_x are the total ion population with ionization state x, U_x is the partition function for the ionization state x, n_e is the number density of electrons, C_I is a universal constant and al other symbols have their ususal meaning. As the ionization likelihood depends on the internal electronic state of the atom the partition function sums up over these different states:

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

where i describes the excitation states,j the ionization states and $g_{i,j}$ the statistical weight of the state i, j. The sum normally diverges slowly so one in practice just sums up until a highly excited state.

The ML MONTE CARLO uses the *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 thus 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:

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

and

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

These approximations do not take into account physical processes like recombination to excited states and the influence of meta-stable levels. They do reproduce the physical state of the plasma relatively well on the other hand and allow for the fast computation of a synthetic spectrum.

1.1.2. Monte Carlo radiative transfer

The ejecta in the ML MONTE CARLO-code is divided into 20 concentric shells with an equal thickness in $1/r$, where r is the radius from the centre of the explosion. These shells have different abundances calculated at each shell midpoint from the well known empirical model W7 (Nomoto et al., 1984). There is a *stratified* version of the code that allows for different abundances in each shell. We have opted to use the simple homogeneous version as the parameter space is very complex even for this homogeneous version. We plan to extend the automatic fitting procedure to use the stratified version at a later date. In addition, we assume a time since explosion t_{exp} from the photometric maximum and a rise time estimate of 19.5 days. The photospheric velocity v_{ph}, the bolometric luminosity $L_{\mathrm{bol}}$and abundances for the chosen elements are the input parameters to the ML MONTE CARLO-code.

The ML MONTE CARLO does not work with individual photons, but with photon packets. Each photon packet, described by frequency and number of photons, and contains the same energy (more photons per packet in the red than in the blue). There are two ways to treat photon interaction. For a pure scattering interaction the photon packet is absorbed at a resonance frequency and then instantaneously reemitted with the same frequency as well as number of photons into a random direction. Newer versions of the ML MONTE CARLO(used in this work), however contain a photon branching implementation. In this case the photon packet is absorbed by the atom and has the chance to be emitted through a different transition. This transition is chosen by a weighted random process. The number of photons in the packet is adjusted to conserve the co-moving energy of the packet. Our one dimensional model is divided into multiple geometric shells. They do have a 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 $B(T)$. An event optical depth is calculated from a uniform random distribution so that $\tau_{\mathrm{event}} = -ln(z); z \in (0, 1]$. There are three possible outcomes for the photon in each Monte Carlo step: 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 $\tau_{\mathrm{event}} = \sigma_{\mathrm{T}} n_e s_e$. Next we calculate the same path length for the lines s_l using as a target opacity $\tau_e + \tau_{\mathrm{line}}$. If both paths are longer than the path to exit the current shell, then the photon exits the current zone and a new Monte Carlo step begins. In the case that 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 we start 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. An initializing Monte-Carlo simulation runs and records each packet status at the mid-point of each shell. This information is used to calculate a new photospheric temperature and a updated plasma conditions (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 photons. Instead we calculate the optical depths from the escaping photons 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) and 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 this by using a spectrum of SN2002bo 10 days before maximum (Benetti et al., 2004). Directly measurable are the redshift of the supernova (and implied distance) and the time of the spectrum relative to maximum. We calculate the time since explosion (t_{exp}) using the photometric maximum and assuming a rise time of 19.5 days. The other parameters are initialized using empirical data (see section **??**).

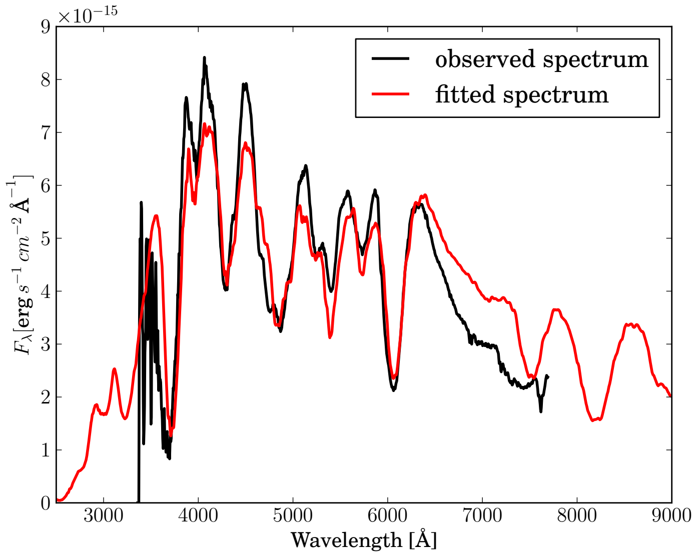

Figure 1.1 Spectrum of SN 2002bo (Benetti et al., 2004) with ML MONTE CARLO-fit by Hachinger (2007). The excess in redwards of 6500 Å is a common problematic features of these fits.

Stephan Hachinger has kindly provided his manually obtained best fitting parameters (for the supernova at this stage (see Figure 1.1). The best fitting parameters for this spectrum are $\log L/L_\odot = 9.05$ and $v_{\text{ph}} = 11700$. We have listed the non-zero abundances in Table

Table 1.1 Abundance parameters for best fit

Atom	Abundance
Carbon	0.08 %
Oxygen	54.9 %
Magnesium	10 %
Silicon	25 %
Sulfur	4.5 %
Calcium	1 %
Titanium	0.01 %
Chromium	0.07 %
primordial Iron	0.07 %
undecayed Nickel	1.5 %

??. Nickel is often given in terms of undecayed nickel (Ni_0) and will then split up into its daughter nuclei ^{56}Coand ^{56}Fe. Fe_0 designates the primordial iron abundance and is added to the iron abundance which results from the nickel decay. We assume in any further discussions in this work that cobalt is always ^{56}Coand is never assumed to be stable.

In Figure 1.1 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 calcium components at the outer edge of the ejecta. This can be often rectified in the stratified version of the ML MONTE CARLO-code. The next major known discrepancy is the excess of flux redwards of ≈ 6200Å. This is a common problem as the wrongful assumption of an underlying black body spectrum overestimates the flux in this region. When fitting manually often one tries to fit the depth of the lines instead of the continuum.

There are three main parameters that have the most influence on the overall spectrum fit: Luminosity, photospheric velocity and abundance in iron group elements. A large offset in L to the best fit parameter is easily visible as a large offset of the continuum (see Figure 1.2. Thus it is easy to constrain the parameter space in L. 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 of atomic lines. In this case however it is hard to measure the photospheric velocity from atomic lines. 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 structure with the given luminosity. A model with a too high photospheric velocity will have expanded more than the observed spectrum and thus will be cooler. This results in a spectrum that is too luminous in the red and not luminous enough in the blue (see Figure 1.3). A secondary effect is that the ion population will be off. This can be determined by judging the fit of SiII lines versus the fit of SiIII lines.

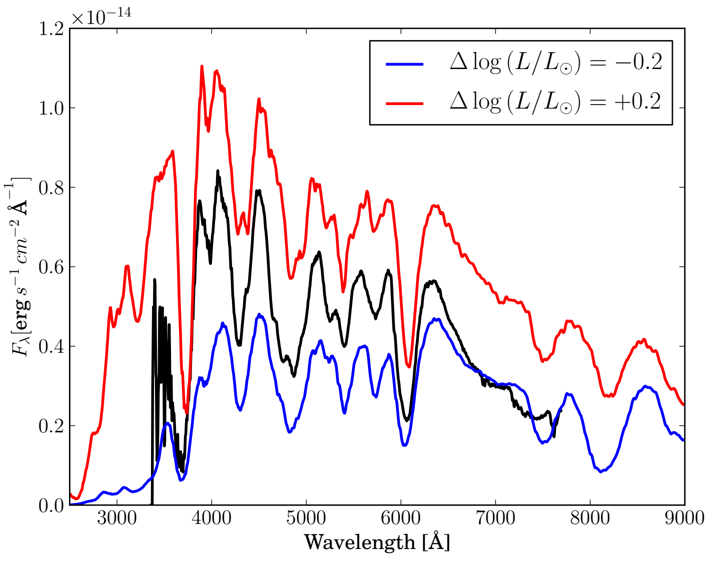

Figure 1.2 We have perturbed the luminosity around the best fit value. The most noticeable effect is the continuum offset. There is also a slight change in the overall slope of the spectrum.

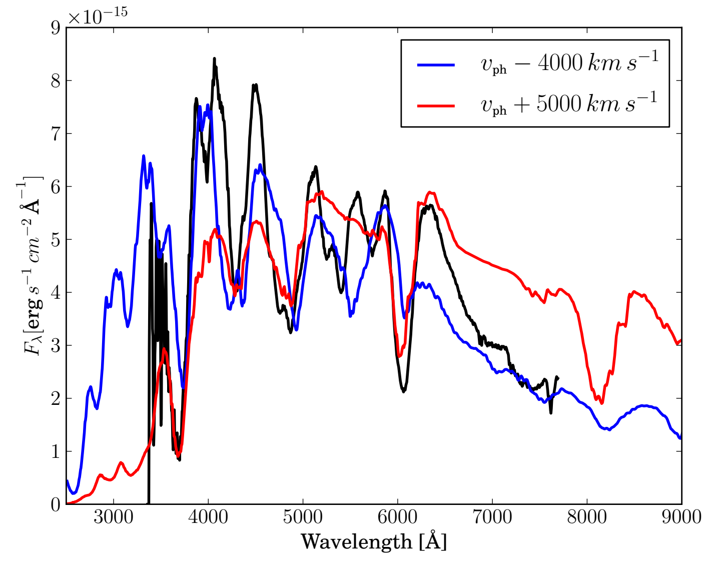

Figure 1.3 The photospheric velocity has been perturbed around the bestfit. The slope of the spectrum changes a too small velocity produces a hot blue spectrum, a too large velocity a cold red spectrum.

The initial abundances for each fit are determined by integrating the abundances above the photosphere in the W7 model (Nomoto et al., 1984).

The IGE have a similar influence on the overall flux distribution as the photospheric velocity. We assume, as discussed previously, no stable Cobalt. The input parameters for Nickel, Cobalt and Iron are the undecayed $^{56}\mathrm{Ni}_0$ and the primordial $^{56}\mathrm{Fe}_0$. We use the time since explosion to calculate the abundances using radioactive decay. The other important IGEs are Titanium and Chromium which have no easily identifiable single lines in the observed spectra. All IGE elements reprocess the flux heavily by absorbing in the bluewards of ≈ 3800 Å and fluoresce in the red part of the spectrum. For example, 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 (mainly influenced by photospheric velocity and luminosity). This near degeneracy causes a very complex parameter space. We disregard the elements Scandium, Vanadium and Manganese as they seem to have little influence on the spectrum at their predicted abundances.

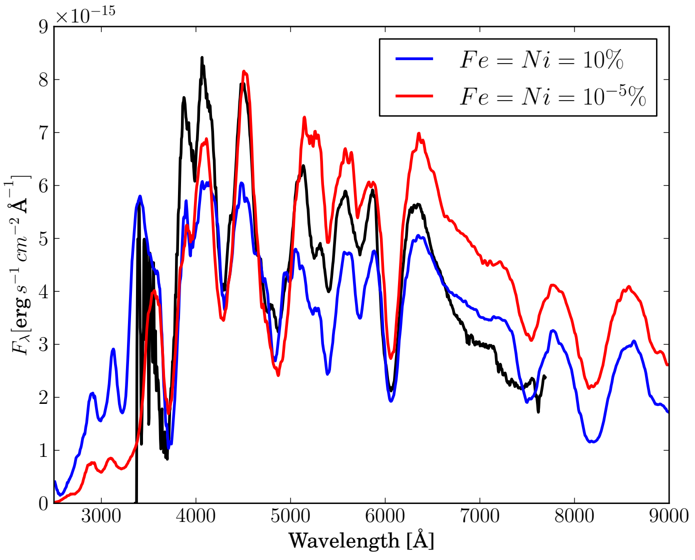

Figure 1.4 When changing the IGE (in this case we have only changed iron and Nickel) the flux is altered in the blue and red part. Too much IGEs and there's not enough flux in the blue and too much flux in the red and vice versa.

There are six other abundances that are taken into account when fitting: Carbon, Oxygen, Magnesium, Silicon, Sulfur and Calcium (see Figure 1.5). Oxygen does not have lines except the Oxygen triplet at 7778 Åand plays a special role in the fitting process. Oxygen acts as a buffer element and is assigned the remaining fraction that is left after all elements have been given abundances. We do acknowledge that this might physically not be correct and a change in the density structure would be more appropriate. This, however would introduce an additional free parameter and make fits much more complex.

Judging the fit of the CaII line at 3700 Åis relatively easy and thus the calcium abundance is usually changed first. Additionally, the choice of temperature imposed by photospheric velocity and luminosity does not have an immense influence on this line. One caveat however is that the CaII line saturates at a certain abundance. If the observed CaII line is close to that limit one can only extract a lower limit for Calcium. Silicon and Sulfur are usually the next elements to be fine-tuned. 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 SiII to SiIII (see Figure 1.5). The strong strong MgII feature near 4300 Åhelps constrains the Magnesium abundance. The weak CII feature at ≈ 6300 Å is the only line to provide constraints for the carbon abundance. In most SNe Ia-spectra this line is weak or not visible.

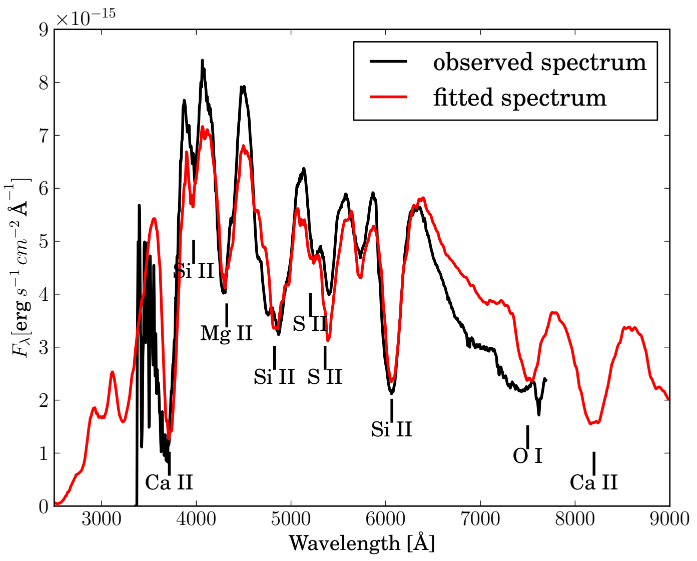

Figure 1.5 We have identified some of the strongest lines that are scrutinized when fitting a SNe Ia spectrum.

The fitting process 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 re-adjust the luminosity, photospheric velocity and IGE. This iterative process continues until a satisfactory fit has been obtained. When closing in on the optimal fit we do observe the dilution factor. Purely theoretical we would expect this to be close to 0.5 for a "physical" fit. We do, however accept values between 0.4 and 0.7 as physical. This large range is accepted due to numerical fluctuation and approximations made by the ML MONTE CARLO. The parameters are often improved further by checking their time evolution through fits of photospheric spectra at different times.

In summary, the fitting of a supernova is a complex procedure and requires a lot of practice. For automating this process we initially tested simple gradient methods. These failed abysmally. We quickly discovered the search space is too complex and evaluation time takes too long (each synthetic spectrum roughly one minute on a modern computer) to use these simple methods. Research in numerical optimization techniques haa made significant process in the last decades. These areas of mathematics have yielded impressive algorithms. Genetic algorithms are easy to implement and are intrinsically parallel. A perfect match for our problem.

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 immense advances in optimization algorithms over the last decades. Among them is the remarkable feat of quickly finding solutions for famous traveling salesman problem with simulated annealing (Kirkpatrick et al., 1983) and later with ant colony optimization (Dorigo & Gambardella, 1997).

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 Holland (1962). These evolutionary algorithms have since become a sizeable subfield of numerical optimization. Genetic algorithms (henceforth GA), a subclass of evolutionary algorithms, were further developed by John Holland's student David Goldberg. (Goldberg, 1989) is the standard textbook for GAs.

Optimization algorithms have two seemingly conflicting goals: exploiting good leads (optimum seeking) while still exploring the parameter space sufficiently. Simple algorithms like Hillclimbing (randomly selecting a point in the neighbourhood of the current point and then picking the more optimal point as the new basis) will exploit good leads but will neglect to explore the search space leading to the convergence on local optima. 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. Which algorithm to use, naturally depends on the type of parameter space. Spaces with independent variables can be solved relatively simply by employing optimum seeking methods. Random search algorithms are suited to parameter spaces with highly co-dependant variables. Genetic algorithms do strike a balance between exploration and exploiting the current best solution. Consequently they can be applied to a wide range of different parameter spaces.

Genetic algorithms are a heuristic search technique to find optimal solutions in n-dimensional search spaces. Generally in optimization we consider the function $f(\vec{x})$ with the multi-dimensional solution $\vec{y}$. In addition, we define a figure-of-merit function $g(\vec{y}) = s$ where s is a scalar. It is the goal to find the input vector $\vec{x}$ to maximize or minimize s. In GAs a single solution ($\vec{x}$) is called an individual (sometimes also referred to as a genome). The individual components of the vector $\vec{x}$ are referred to as the genes. We also differentiate between the vector representation called genotype, and the solution vector for an individual ($\vec{y}$ using the previous notation) called phenotype. This is similar to biology where the input $\vec{x}$ can be thought of as the DNA sequence. The phenotype in biology however does not resemble a vector. Mathematically speaking it is possible to have multiple genotypes map to one phenotype, however each genotype only maps to one phenotype. The figure-of-merit function $g(\vec{y}) = s$ is often referred to as the fitness function. 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 (also called generation). 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 (or generation) $P(t+1)$ is formed by choosing, in the *select step*, the more fit individuals. Some or all of the new population undergo transformations in the *recombine step*. Generically these transformations are called genetic operators. We define 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 (perform the computation $g(f(\vec{x}_i)) = s_i$) 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

```
$t \leftarrow 0$
initialize $P(t)$
evaluate $P(t)$
while (not termination condition) do
  $t \leftarrow t + 1$
  select $P(t)$ from $P(t-1)$
  recombine $P(t)$
  evaluate $P(t)$
end while
```

An optimization problem needs to have the following traits to be solvable by a GA:

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

If the problem fulfills all these requirements one can start constructing a GA. This involves multiple steps the first of which is choosing a suitable genetic representation for each solution in the parameter 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 only offers one gene, whereas binary encoding, depending on the requested precision of the value, offers multiple genes. This becomes obvious in the one-dimensional minimization xample: $f(\vec{x}) = (x_0 - 3.141)^2$. The solution vector that minimizes the problem in value encoding is $\vec{x} = (3.141)$ using IEEE 754 floating point encoding the optimal vector is $\vec{x} = (0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,1,0,0,0,1,0,0,1,0,1)$. 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 genetic operators like mutation) can dramatically change the encoded value (e.g. Chakraborty & Janikow, 2003). This can improve covering of search space but also can hinder the code from converging. When using binary encoding for many input variables the genomes can get incredibly long. GAs have been shown to perform poorly for very long genomes. 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; Goldberg, 1990)

Finally, an important encoding to mention 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 1.6).

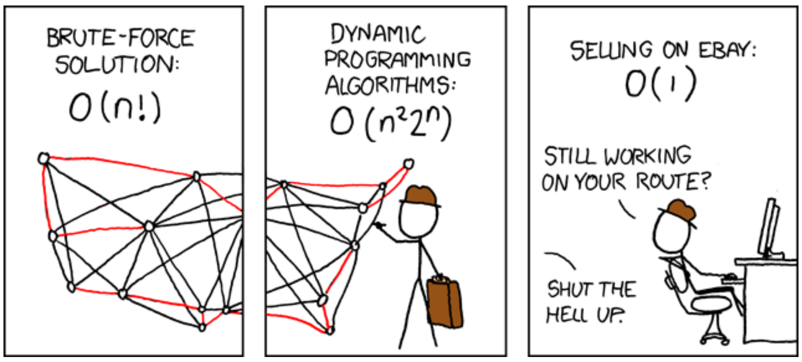

Figure 1.6 The death of the TSP with the advent of online sales. (reproduced with kind permission by xkcd.com)

The fitness function maps the phenotype ($\vec{y}$) to a scalar. Fitness functions are very problem dependent and are often hard to define. It might not always be possible to find a function that maps desirability to exactley one value (e.g. it is not possible to map desirable traits of a car to one number and optimize that). This class of problem is generally referred to as multi-objective optimization and is an active field in GA-research (Konak et al., 2006). Describing this subfield is outside the scope of this work. A GA is required to follow promising leads but also explore the parameter space. It sometimes happens that a fitness function values a small fraction of individuals extremely high when compared to the rest of the solutions. This can become a problem when this high fitness value is only tied to a few genes. For example, a right value of the first gene could cause much higher fitness then correct values of other genes in a single individual. The subsequent populations will then be reigned by the genes of this one individual which prohibits the GA to explore the search space. In GA-terms individuals with extremely high fitness based on very few gebes are called "superindividuals". They can drastically reduce the genetic breadth and often cause the GAto fail. One way to overcome this problem is scaling the fitness of all individuals after they have been computed. There are other steps which can be undertaken in the *select step* described later. The scaling is often a simple algebraic transformation, like linear or exponential scaling. The coefficients for this transformation are often calculated by using the average, minimal and maximal value of the fitness distribution. A specific example can be found 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 **?**.

The most basic quantity to consider when choosing the initial population is that of population size. Generally the population size remains the same over the course of a GA. The population size should be chosen in relation to the size of the parameter space. A small population size and a large search space can lead the GA to find local optima rather than the global optimum. We have in our work chosen a population which is roughly 15 times bigger than the number of genes. After having chosen the population size the most basic method of selecting the initial population is to draw individuals uniformly and randomly from the entire search space. One might however know a probability distribution for the parameter space and can draw randomly weighted by the distribution (e.g. when trying to find find parameters for a random star, we can rule a $20\,M_{\odot}$white dwarf with some likelihood). An initial population that is closer to predicted optimal value will converge faster, but won't explore the parameter space that well.

Once we have evaluated the fitness for each member of the individual population the next step is the *selection step*. 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 *elitism* in which a fraction of the fittest individuals is selected to advance unaltered to the next generation. On the other hand one can discard a fraction of the least fit individuals. These discarded individuals (including their gene-combinations) 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 in 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). The most widely used of the selection algorithms is *roulette wheel selection* (see Figure 1.7). *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 1.8). The previous described fitness scaling and rank selection have similar effects. They do alleviate the problem of "super-individuals" that will dominate the following generations and often lead to the failure of the GA. Finally, 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 (also known as the next generation). 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.

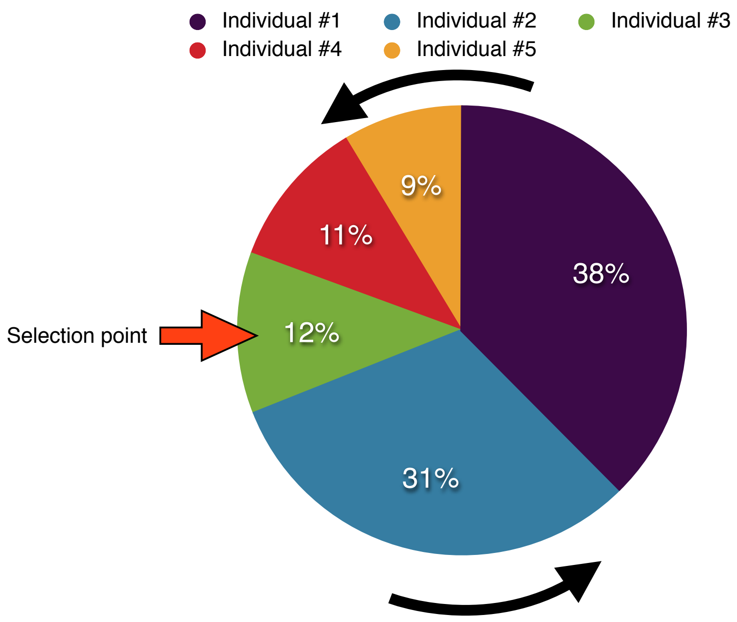

Figure 1.7 The individual fitnesses are assigned proportional fractions on the roulette wheel. The wheel is then spun and will slowly decelerate and stop at some point. Individuals with a higher fitness have a higher chance of being chosen with this method.

To describe these two options in detail we assume a mating pool with two individuals. The simplest form of a crossover is the single-point crossover (see Figure 1.9). 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 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. Multi-point (see Figure 1.9) is essentially just a extrapolation of two point crossover. Finally, there is uniform cross-over in the case that each gene is selected randomly with equal chance from either parent. 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 they are 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.

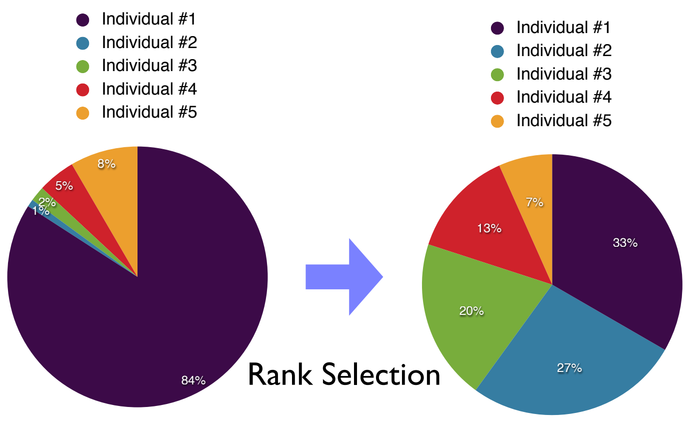

Figure 1.8 We assign new fitness values to individuals before assigning them probabilities on the roulette wheel. The fitnesses are assigned by location in an ordered list. The least fit individual gets assigned the value 1 the fittest individual the number n, where n is the population size. This is can be viewed as a special case of fitness scaling. After the new fitnesses have been chosen we use normal roulette wheel selection to select for the mating pool in the population.

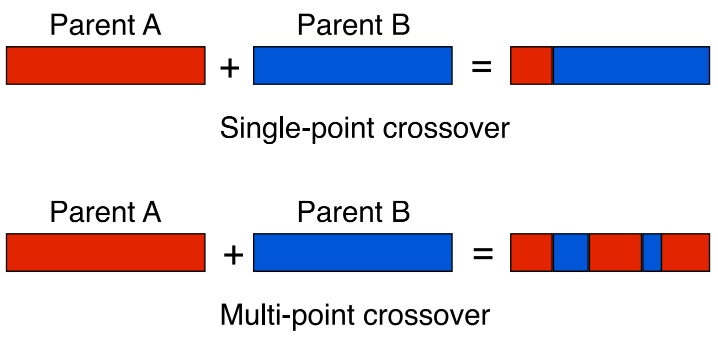

Figure 1.9 In the single crossover a random point in the Individual is chosen. Before this point the genes are taken from the first parent and after that the we use the genes from the second parent. Using the same random number it easily allows for the creation of two children by reversing the roles of first and second parent. The multi-point crossover employs multiple places in the genome where the crossover happens.

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 the 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 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 many GA-implementations 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. A feature that separates GAs from traditional optimization algorithms is that 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. Many other search algorithms require the evaluation of one solution to calculate the next step. The inherent parallel nature of GAs makes it easy to explore large search spaces in the era of increasingly parallel computing.

There are many advantages to using GAs, however 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 a pattern matching schemata were the $*$ symbol can be thought of as *don't care*-symbol. The schemata $0, 1, 0, *$ for example, 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. The given example is an third order schemata. Schematas provide a powerful way to describe similarities in a set of genomes. A whole population of solutions therefore samples a range of different schematas . Essentially low-order and above-average schemata receive exponentially increasing trials in subsequent generations of a GA. 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

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_{ph}), Carbon, Magnesium ,Silicon, Sulfur, Calcium, Chromium, initial Ni (Ni_0) and primordial iron (Fe_0). As in the manual fitting example the time since explosion as well as the luminosity distance (for scaling purposes) are given. The evaluation for each spectrum takes roughly one minute on a modern computer which 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 taken by Hachinger (2007) and (Hachinger, 2011). It took relatively long and would not converge. We realized quickly the search space is far to complicated for such simple methods. In addition, we were limited to one processor with this gradient method as the ML MONTE CARLO itself is not parallel. GAs seem the perfect choice for this problem. They are easy to implement, can easily be parallelised and are relatively immune to local minima. For most of our optimization tests we worked with the supernova SN 2002bo (Benetti et al., 2004). This supernova is believed to be a “Branch”-normal and is spectroscopically well sampled. For most of our tests we worked on the t_{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 these where applicable.

1.4.1. 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_{ph}(within some bounds). This method does not work for the elemental abundances as the sum of the abundances needs to add up to 1. For many normal supernovae the W7-abundances (Nomoto et al., 1984) give a good starting point. A population that is distributed around the optimum value converges mucg quicker than a uniformly sampled one. To calculate these abundances we need to know the photospheric velocity. Using data from Benetti et al. (2005) 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.

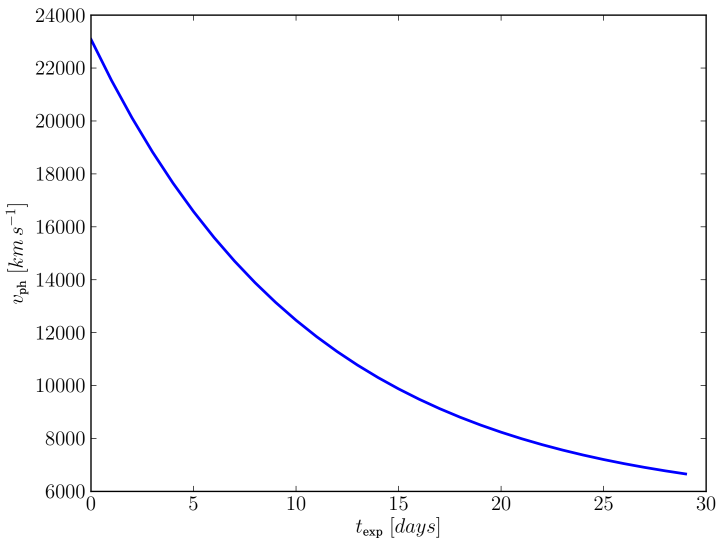

Figure 1.10 Estimated initial guess for photospheric velocity against days after explosion.

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 abundance fractions.

When creating our initial population we draw uniformly random for a preselected luminosity range (for the moment determined manually), a $v_{\rm ph}$-range 4000 km s^{-1}above and below the $v_{\rm ph}$-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 element to be drawn would always have a very low chance of obtaining a high abundance value. Thus we randomize the order of drawing the elemental 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(e.g. Iwamoto et al., 1999).

- 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 is 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 spectrum synthesizers.

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 1.1). We still regard the line depth and line shape to be correct. 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 tried multiple approaches. At first we tried to de-weight the fit in the problematic region. This artificially introduced weighting factor introduces another parameter which might have to change it for the GA to succeed on different spectra. This would defeat the point of an automatic fit. As a second approach we tried to fit and subtract the continuum in the problematic region before creating our fitness figure. In addition, the ML MONTE CARLO supllies us with an estimate of the dilution factor W at the photosphere. This factor is expected to be close to 0.5 (see description in **??**). Hachinger (2007) and Hachinger (2011) have found that in all cases a good fit will have a value of the dilution factor between 0.4 and 0.7. We have incorporated this into the fitness calculation and de-weight spectra with a value outside this range to a great extent. We should mention that we have tried, in addition to calculating the root-mean-square of the spectra, a number of different methods. Most notably we tried to use neural-networks to perform a goodness of fit analysis. We however abandoned this effort as the training set to calibrate neural networks requires a large number of well fitted spectra which are not available.

Once the fitness is calculated for each individual we use the method of fitness scaling to preempt premature convergence in early generations (caused by the so called "super-individuals") as well as creating a steeper fitness gradient in later generations. We have decided to use a linear fitness scaling as described in Goldberg (1989):

$$f' = af + b,$$

where $f\prime$ designates the scaled fitness and f the raw fitness. In all cases we want to make sure that the average of the scaled fitness f'_{avg} equals that of the average of the raw fitness f_{avg}. We will first try to find a linear relation so that the new maximum fitness f'_{max} is C_{mult} times the average fitness (f_{avg}). We choose $C_{\mathrm{mult}} = 2$ as suggested by Goldberg (1989). This operation will scale early "super-individuals" down and the rest of the population up and preempts premature convergence. In the later phases of the GA, when the fittest individuals and the bulk of the population have similar fitness values, this operation would lead to negative fitnesses for some individuals. In that case we find scaling parameters that maps the least fit individual to a fitness value of 0, while still maintaining $f'_{\mathrm{avg}} = f_{\mathrm{avg}}$. Once we have scaled the fitnesses we move to the selection process.

In the current version of the Dalek Code we employ *elitism* (10% of fittest individual advance immediately to next generation). We have also experimented with other modifications to the mating population, but found them to be subpar. The mating pool we have chosen has only two slots. For now we select the individuals for the mating pool using the standard *roulette wheel selection*. We use a crossover probability of 90%, this means that in 10% of the cases we do not perform crossover but the child is a copy of the first parent. If crossover occurs, we perform a uniform crossover as we felt that we do not want the ordering of the parameters to matter. We have also experimented with arithmetic crossovers by taking the mean of parents. This meant that often useful traits would entirely disappear and we switched back to uniform crossovers. We will experiment in the near future with different crossover techniques like single-point crossovers. The new individual, created by crossover, is subjected to possible mutations before being placed into the new population. The Dalek Code employs different mutation strategies for the different parameters. The abundances are just multiplied by a uniform random number (we have tried gaussian random numbers) with a relatively high chance of currently 7%. As luminosity and photospheric velocity are physically different we treat them differently to the abundances. In both cases we add a uniform random number, in addition to having different mutation probabilities. After the mutation step we see if the buffer element oxygen has a negative abundance. A negative abundance implies that the other elements add up to more than 1. If oxygen has positive abundance the code goes on to check the abundance ratios. If the child passes both these tests it is placed in the new population, if not it is discarded. This process is repeated until the size of the new population (including the members of the old population that advanced through *elitism*) reaches 150.

This process of selection, recombination and mutation is repeated over many generations. We have experimented with up to a 1000 populations in one run. Our scheduler can achieve a throughput of one generation per minute (in optimal conditions where all CPUs are free). Our principal spectrum for the testing was SN 2002bo 10.4 days post explosion (see Figure 1.1). In the presented case we let the GA run for 188 generations. Figure 1.11 shows how the genetic algorithm over many generations improves the fitness of the indivduals.

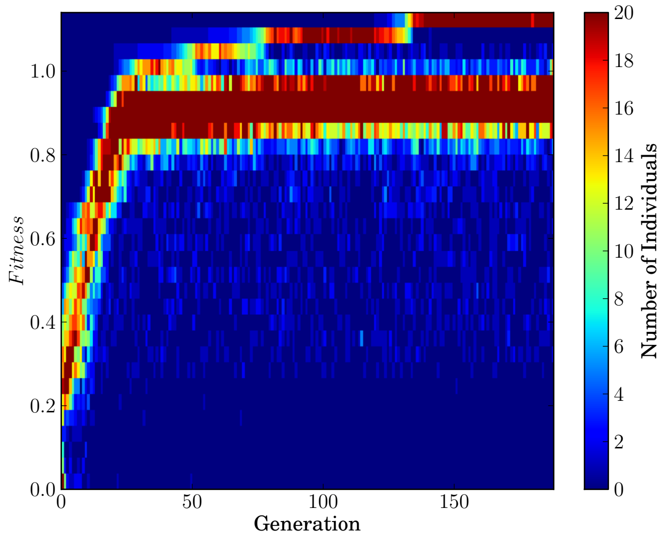

Figure 1.11 Evolution of fitness of in the generations. In later generations there is a clear divide between the members that have advanced through elitism and the bulk of the population. This suggests a too high mutationrate or the convergence of the algorithm.

The step-pattern is very common in genetic algorithms: One individual has a favorable mutation and it takes several generations for the others to catch up. During the last generations one can clearly see that there is a gap between bulk of the population is and the top 10%. This gap is caused by the contrast of relatively high mutation in the main individuals against the mutation free advancement of the top 10% (*elitism*).

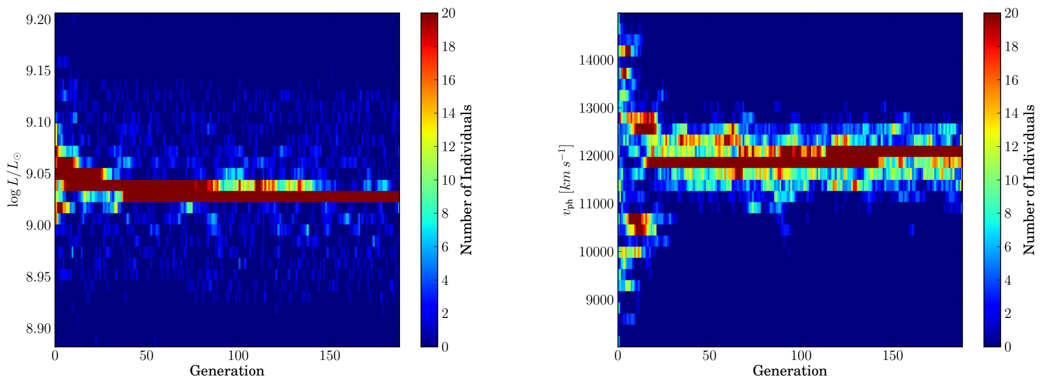

Figure 1.12 The evolution of both luminosity and photospheric velocity shows that even at late phases of the algorithm there are still new combination that are being trialled. The very quick convergence in both cases however is a bit worrisome and will be analysed in future experiments.

Figure 1.12 gives a good overview of how the genetic algorithm process through the search space. In our example the algorithm converges relatively fast (after roughly 30 generations). Even though there is a "main"-population the algorithm still tries out different combinations. Currently we still have the problem that for certain initial random seeds the algorithm does not find the global optimum but converges prematurely on a local optimum. We believe the quick convergence seen in Figure 1.12 might be evidence for this. This again is an area of active research and two experts in evolutionary optimization joined our team recently to address this problem.

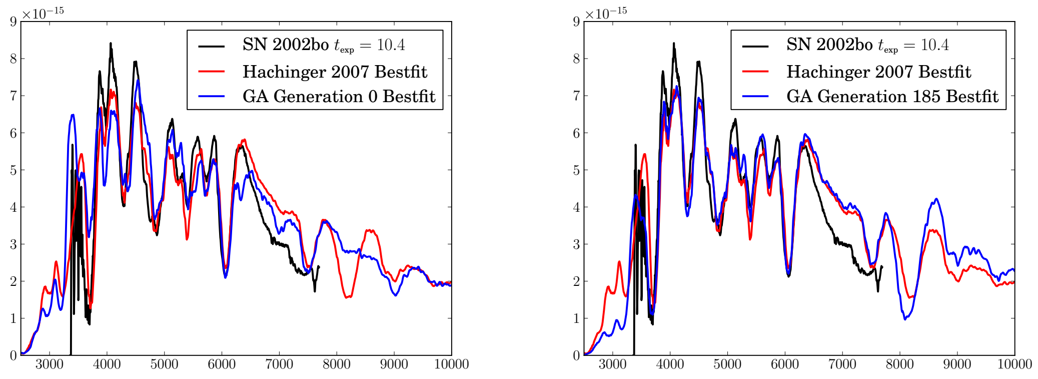

Figure 1.13 The best individual of the first generation (left) and the best individual after 185 generations (right). This initial result demonstrates that GAs are able to conquer this problem. A more stable convergence and thorough exploration of the parameter space is however necessary to use this technique on a set of supernova spectra.

Although there are still many outstanding problems we can show that the GAs are able to solve the SNe Ia fitting problem. In Figure **??** we show the best fit in the first generation as well as the best-fit in the last generation. As comparison we show the best fit obtained by Hachinger (2007). The GA has been successful in reproducing the main features of the SNe Ia-spectrum. The de-weighting in the infrared is working very well. We are currently trialing the current code on different spectra with mixed results.

1.5. Conclusion

GAs are powerful tools, which can easily search a vast parameter space in parallel and avoid local optima. The intrinsic parallel structure has been a tremendous advantage in our experiments. We have shown that GAs can successfully solve the problem and fit a SNe Ia-spectrum. We do acknowledge that our current implementation does not reliably work on all spectra and there remains lots of fine-tuning work to be done. This tweaking is very common in GA implementations, but often requires experts in numerical optimization for speedy advancement. James Turner and Irene Mosner are experts in evolutionary optimization (their research focuses on differential evolutionary algorithms) and are currently exploring with our help the search space. On a different front we are currently exploring to optimize the exploration speed. Even on current high-end machines the ML MONTE CARLO takes one minute per synthetic spectrum per CPU. One of the ideas is creating a parameter grid around all or some of our search area. We could then use fast and efficient interpolation techniques (one of these is described in Chapter **??**) and explore different settings for our genetic algorithm much more quickly.

Over the course of this project we have realized that the field of numerical optimization has made huge strides in the last decades. Most of the new algorithms are not well known in astronomy and are used even rarer. On the other hand experts in the field of numerical optimization are often in desperate need for "real world" applications. They are more than willing to apply their knowledge to problems in our field. We strongly believe that there is an important ground for collaboration that would benefit both sides immensely.

BIBLIOGRAPHY

Benetti, S., Cappellaro, E., Mazzali, P. A., Turatto, M., Altavilla, G., Bufano, F., Elias-Rosa, N., Kotak, R., Pignata, G., Salvo, M., & Stanishev, V. 2005, ApJ, 623, 1011

Benetti, S., Meikle, P., Stehle, M., Altavilla, G., Desidera, S., Folatelli, G., Goobar, A., Mattila, S., Mendez, J., Navasardyan, H., Pastorello, A., Patat, F., Riello, M., Ruiz-Lapuente, P., Tsvetkov, D., Turatto, M., Mazzali, P., & Hillebrandt, W. 2004, MNRAS, 348, 261

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

Dorigo, M. & Gambardella, L. M. 1997, Ant colonies for the travelling salesman problem

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]

—. 1990, Complex Systems, 5, 139

Hachinger, S. 2007, Master's thesis, TU München

—. 2011, PhD thesis, TU München

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)

Iwamoto, K., Brachwitz, F., Nomoto, K., Kishimoto, N., Umeda, H., Hix, W. R., & Thielemann, F.-K. 1999, ApJS, 125, 439

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

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