Portfolioselektion durch Simulated Annealing mit SAS

Gregor Pfletschinger
Information Works GmbH
Rolshoverstr. 45
51105 Köln
g.pfletschinger@
information-works.de

Peter von Tessin
Information Works GmbH
Königstr. 10c
70173 Stuttgart
p.vontessin@
information-works.de

Zusammenfassung

Portfolioselektion mit diskreten Investitionsobjekten ist eines der klassischen Anwendungsgebiete heuristischer Optimierungsverfahren in der Finanzbranche. Die Zusammensetzung eines solchen Portfolios muss in Hinblick auf verschiedene Kriterien optimiert werden. Wir stellen eine Implementierung des Simulated Annealing Algorithmus in SAS zur optimalen Auswahl von Krediten für ein Verbriefungsportfolio vor. Der Schwerpunkt der Darstellung liegt auf der effizienten Implementierung eines solchen Algorithmus in SAS.

Schlüsselwörter: Optimization, Simulated Annealing, SAS, Portfolio Selection, Securitization

1 Einleitung

Die Verbriefung von Krediten ist mittlerweile ein fester Bestandteil der Finanzierung von Konsumgütern. Wenn Verbriefung, dank extrem gewagter Portfoliokonstruktionen, seit der Finanzkrise auch einen etwas belasteten Ruf haben mag, so ist das moderne Kreditgeschäft dennoch kaum noch ohne sie vorstellbar. Und besonders vor diesem Hintergrund ist es natürlich wichtig, die Zusammensetzung eines Portfolios möglichst optimal an den von möglichen Käufern verlangten Kriterien auszurichten.

Das Problem der Portfolioselektion für die Kreditverbriefung ist insofern diskreter Natur, als ein Kredit entweder in dem zu verbriefenden Portfolio enthalten sein kann oder nicht. Aus diesem Grund eignet sich ein heuristischer Optimierungsalgorithmus wie Simulated Annealing auch besonders gut, um die möglichst optimale Zusammensetzung des Portfolios zu ermitteln. Die sonst übliche Vorgehensweise, einen Simplex Algorithmus zur Bestimmung der optimalen Portfoliozusammensetzung zu verwenden (siehe beispielsweise [1]), ist dagegen weniger geeignet, da die einzelnen Kredite nicht in stetigen Anteilen dem Portfolio zugefügt werden können, sondern entweder darin enthalten sind oder nicht.

Doch während der Implementierung des Simulated Annealing Algorithmus mussten wir feststellen, dass es die für einen solchen Algorithmus in SAS benötigte Laufzeit ist, die uns die größten Probleme bereitet. Damit zumindest annähernd optimale Portfolien er-

stellt werden können, bedurfte es anfänglich einer Laufzeit, die um ein vielfaches höher als ursprünglich geplant war und die von unserem Kunden gemachten Parameter-Vorgaben eindeutig verletzte. Aus diesem Grund waren wir gezwungen, während des Projekts das Design des Algorithmus noch einmal so zu verändern, dass auch in einer wesentlich kürzeren Laufzeit nahezu optimale Lösungen erreichbar werden.

Eine Darstellung der in diesem Zusammenhang gewonnenen Erkenntnisse ist zentraler Gegenstand der vorliegenden Arbeit.

2 Das grundlegende Design des Algorithmus

2.1 Simulated Annealing

Die Funktionsweise eines Simulated Annealing Algorithmus dürfte allgemein bekannt sein. Ausgehend von einer initialen Lösung wird diese immer wieder in kleinen Schritten verändert, und es wird überprüft, ob diese Änderung zu einer Verbesserung der „Fitness“, zu einer Annäherung an den durch die Kriterien bestimmten Optimalzustand, geführt hat. Ist dies der Fall, so wird eine solche Änderung beibehalten. Sollte es zu einer Verschlechterung der Fitness gekommen sein, so wird die Änderung im Allgemeinen wieder rückgängig gemacht. Aber in manchen Fällen wird auch eine Änderung, welche zu einer Verschlechterung der Fitness führt, beibehalten, um auf diese Weise sicher zu stellen, dass man nicht auf das nächstbeste lokale Optimum zusteuert, sondern nach Möglichkeit das globale Optimum findet. Die Funktion, welche bestimmt, dass auch eine die Fitness verschlechternde Änderung beibehalten wird, ist eine von der „Temperatur“ abhängige Zufallsvariable. Als Temperatur wird ein Parameter bezeichnet, der während des gesamten Laufs des Algorithmus in jeder Iteration ein wenig verringert wird. Dies hat zur Folge, dass die Wahrscheinlichkeit eine verschlechternde Änderung anzunehmen, über die Laufzeit des Algorithmus immer geringer wird. Nach vollzogener Temperaturabsenkung wird noch eine gewisse Zeit nachoptimiert. Siehe [2] für eine allgemeine Darstellung des Simulated Annealing Algorithmus.

2.2 Simulated Annealing zur Optimierung eines Verbriefungsportfolios für Konsumentenkredite

In dem uns vorliegenden Fall der Erstellung eines Portfolios für die Verbriefung von Konsumentenkrediten lässt sich die Arbeitsweise des Algorithmus folgendermaßen beschreiben. Wir beginnen mit einer Kandidatenliste, die alle Kredite enthält, die entsprechend gewissen Kriterien überhaupt zur Verbriefung zugelassen sind. Aus dieser Kandidatenliste wählen wir rein zufällig ein erstes Portfolio aus, welches genau so viele Kredite enthält, dass wir das wichtigste Kriterium, nämlich den angestrebten Wert des Verbriefungsportfolios, annähernd exakt erreichen. Die Fitness dieses Portfolios entspricht der Summe der Differenzen des Erfüllungsgrades jedes einzelnen Kriteriums zum gewünschten Optimalwert. In jeder Iteration werden dann zufällig ein oder mehrere Kredite, die bereits im Portfolio sind, ausgewählt und aus dem Portfolio entfernt. Eben-

falls werden ein Kredit oder mehrere, die momentan noch nicht im Portfolio sind, ausgewählt und in das Portfolio übernommen. Erneut wird die Fitness berechnet. Hat diese sich verbessert (die Differenz zu den als optimal angesehenen Kriterienerfüllungsgraden hat abgenommen), so wird die Veränderung beibehalten. Hat diese sich verschlechtert, so wird die Änderung wieder rückgängig gemacht, außer der mit Hilfe der Temperatur berechnete Parameter erlaubt die Beibehaltung auch dieser die Fitness verschlechternde Veränderung. Dieses Vorgehen wird dann solange wiederholt, bis eine die Kriterien möglichst optimal erfüllende Portfoliozusammensetzung erreicht wird.

2.3 Zentrale Bestandteile des Algorithmus

Demzufolge sind die wichtigen Bestandteile des Algorithmus:

- Kandidatenliste: alle zur Verbriefung geeigneten Kredite.
- Portfolio: die zur Verbriefung ausgewählten Kredite.
- Die Kriterien, welche das Portfolio zu einem als optimal festgelegten Grad erfüllen muss.
- Fitnessberechnung: die Funktion zur Berechnung der Differenz zwischen dem durch das Portfolio gegebenen Erfüllungsgrad der einzelnen Kriterien und dem als optimal angesehenen Erfüllungsgrad der einzelnen Kriterien.
- Kreditauswahl Methode, die es ermöglicht, zufällig einen oder mehrere Kredite entweder im Portfolio oder in der Kandidatenliste auszuwählen.
- Akzeptanzfunktion, welche unter Verwendung der Temperatur bestimmt, ob eine die Fitness der Lösung verschlechternde Situation trotzdem beibehalten wird.

Das wichtigste der von dem Portfolio zu erfüllenden Kriterien ist der Gesamtwert der im Portfolio enthaltenen Kredite. Dieser Portfoliowert p_p muss dem als optimal angesehenen Zielwert p_z so gut wie möglich entsprechen. Das zweite Kriterium ist die durch das Portfolio ermöglichte Verzinsung des eingesetzten Kapitals, welche sich durch den gewichteten Durchschnitt der Zinssätze der einzelnen Kredite berechnen lässt. Diese Verzinsung des Portfolios r_p darf nicht geringer als ein vorher festgelegter Minimalzinssatz r_m sein. Weiterhin existiert eine Anzahl *n* von Kriterien, die sich auf qualitative Eigenschaften der einzelnen Kredite beziehen. So könnte beispielsweise gewünscht sein, dass nur eine gewisse Anzahl von Automobil- oder Motorradkrediten in dem zu verbriefenden Portfolio enthalten sind. Diese Anteile der gewünschten Art von Krediten im Portfolio bezeichnen wir mit c_p^i , i=1,…n die wieder einen vorgegebenen Anteil c_z^i nicht überschreiten dürfen.

Damit hat die Fitness-Funktion (oder besser die Handicap-Funktion, da wir die angegebene Funktion ja minimieren wollen) die folgende Form:

Wobei $$f(p,r,c) = \frac{p_p - p_z}{p_z} + I_{r_p < r_z} \frac{r_p - r_m}{r_m} + \sum_{i=1}^{n} I_{c_p^i > c_z^i} \frac{c_p^i - c_z^i}{c_z^i}$$

$$I_{x>y} = \begin{cases} 1 & \text{wenn} \quad x > y \\ 0 & \text{sonst} \end{cases}$$

3 Implementierungsdetails

3.1 Naive Implementierung mit zwei Tabellen

Entsprechend dem im vorangegangenen Abschnitt dargestellten grundlegenden Design für einen Simulated Annealing Algorithmus begannen wir die Implementierung mit zwei getrennten SAS Tabellen für das Portfolio und die Kandidatenliste. In jedem Iterationsschritt wählten wir dann einen oder mehrere Kredite aus, um diese in die jeweils andere Tabelle zu übernehmen. Nach dieser lokalen Änderung wird erneut die Fitness mit Hilfe des folgenden Makros berechnet:

```
/************************ermittleFitness*************************/
/* Macro zur Ermittlung der Fitness                             */
/* Parameter: typ ist Typ der Implementierung, tabName ist der  */
/* Tabellenname, fitName ist der Name der resultierenden        */
/* MacroVariablen, anzName ist der Name der Mac-Variablen für   */
/* die Anzahl und whereBedingung ist die u.U. notwendige        */
/* whereBedingung                                               */
/****************************************************************/

%macro ermittleFitness(typ, tabName, fitName, anzName,
          whereBedingung);

     %global &fitName. &anzName.;

     /* Werte für Portfolio ermitteln */
     proc sql noprint;
          create table PfWerteAktuell as
               select    "&typ." as typ, &i. as iteration,
               &temp. as Temperatur ,
               put(&anzrein.,z3.)!!"-"!!put(&anzraus.,z3.) as
               reinRaus,
               count(*) as PfAnzahl, sum(saldo) as PfSaldo,
               mean(kriterium_1) as PfKrit1,
               mean(kriterium_2) as PfKrit2,
               mean(kriterium_3) as PfKrit3,
               mean(effektivVerzinsung) as PfZins,
               abs(calculated PfSaldo-&SaldoZiel.)/&saldoZiel. as
               SaldoVal,
               abs(calculated PfKrit1-krit1Ziel.)/&krit1Ziel. as
               krit1Val,
               abs(calculated PfKrit2-krit2Ziel.)/&krit2Ziel. as
               krit2Val,
```

```
            abs(calculated PfKrit3-krit3Ziel.)/&krit3Ziel. as
            krit3Val,
            -(calculated PfZins-&ZinsZiel.)/&ZinsZiel. as ZinsVal,
            calculated SaldoVal + calculated krit1Val + calculated
            krit2Val + calculated krit3Val + calculated ZinsVal as
            fitness,
            datetime() - &startPunkt. as zeit
        from &tabName.
        &whereBedingung.;

        select fitness into : &fitName. from PfWerteAktuell;
        select PfAnzahl into :&anzName. from PfWerteAktuell;
    quit;
%mend;
```

Sobald die Fitness der neuen Auswahl berechnet ist, wird mit Hilfe des folgenden SAS Codes herausgefunden, ob sich die Fitness verbessert hat. Sollte dem nicht so sein, wird mit Hilfe einer von der Temperatur abhängigen Zufallsvariable festgelegt, ob die neue Auswahl nicht trotzdem angenommen werden sollte:

```
/* Differenz der Fitness-Werte: bisheriger Wert minus          */
/* Herausforderer-Wert. Also ist der Wert positiv, wenn der    */
/* neue Wert besser als der alte ist. Je größer ein negativer  */
/* Wert ist, umso schlechter der Herausforderer.               */

%let fitnessDifferenz = %sysevalf(&fitnessBisher.
    &fitnessHerausforderer.);

/* Jetzt Wahrscheinlichkeit berechnen, ob Änderung angenommen  */
/* wird, obwohl Herausforderer schlechter ist als bisheriger   */
/* Wert. Dazu zunächst prüfen, ob wir nicht schon auf          */
/* minTemperatur abgekühlt haben und ohne Annealing-Zusatz     */
/* laufen ...                                                  */

%if %sysevalf(&temp. gt &minTemp.) %then
%do;

/* Zur Sicherheit nochmal den null-Fall abfangen, damit keine  */
/* division by zero auftritt.                                  */
/* Ansonsten folgt hier die verwendete Gleichung für das       */
/* Simulated Annealing (e hoch Delta/Temperatur)               */

%if &temp. ne 0 %then %let prob =

    %sysevalf(%sysfunc(exp(1000*(&fitnessDifferenz.)/&temp.)));
%else %let prob = 0;
    %let temp = %sysevalf(&temp. - &tempStep.);
%end;
%else %let prob = 0;

%let trotzdem = 0;
```

```
%let zufall = %sysfunc(ranuni(-1));
%if %sysevalf(&zufall. lt &prob.) %then %let trotzdem = 1;
```

Allerdings stellte sich relativ bald heraus, dass eine solche Implementierung mit zwei SAS Tabellen, zwischen denen einzelne Kredite in jedem Iterationsschritt ausgetauscht werden, ausgesprochen langsam ist, in jedem Fall viel zu langsam für die im Konsumentenkreditgeschäft allgemein übliche, ausgesprochen hohe Taktfrequenz.

3.2 Implementierung mit einer einzigen Tabelle

Um den Effizienzverlust, der mit dem Austauschen einzelner Kredite zwischen den beiden SAS Tabellen einhergeht, nach Möglichkeit einzusparen, verwenden wir nur noch eine einzige Tabelle, in der alle zur Verbriefung in Frage kommenden Kredite enthalten sind. Diese eine Tabelle enthält dann eine Indikatorvariable, die anzeigt, ob der jeweilige Kredit im Portfolio enthalten ist oder nicht. Auf diese Art ist es uns möglich, einen Index über das gesamte Dataset zu legen, der alle weiteren Operationen wesentlich beschleunigt.

Allerdings macht diese Art der Implementierung eine wesentlich ausgereiftere Vorgehensweise bei der Auswahl der für eine lokale Veränderung zu verwendenden Kredite notwendig. Im Fall der naiven Implementierung mit zwei Tabellen war es möglich, zur Auswahl einzelner Kredite einfach die Liste von oben nach unten durchzugehen und mit einer gewissen (recht kleinen, in einem gewissen Verhältnis zur Länge der Liste stehenden) Wahrscheinlichkeit einzelne Kredite auszuwählen. Dieses Vorgehen ist im Fall einer einzigen Tabelle, in der sowohl Kredite aus der Kandidatenliste als auch die bereits für das Portfolio ausgewählten Kredite enthalten sind, nicht mehr direkt anwendbar. Die Auswahlwahrscheinlichkeit der einzelnen Kredite müsste dann von ihrem jeweiligen Status abhängig sein. Aus diesem Grund und um eine noch bessere Kontrolle über den Auswahlmechanismus zu erhalten, haben wir uns entschieden, die auszuwählenden Kredite in einem vorherigen Schritt mit Hilfe von Zufallszahlen zu ermitteln. Dieses Vorgehen ist im folgenden SAS Code implementiert:

```
/************´************zufallszahlen************************/
/* Macro zur Generierung einer Zufallszahlen-Liste             */
/* Parameter: Obergrenze für den Wert, Anzahl der Werte und    */
/* Listenname. Es werden also <randAnzahl> Zufallszahlen       */
/* zwischen 1 und <randMax> gezogen und in die Macrovariable   */
/* <liste> geschrieben.                                        */
/***************************************************************/
%macro zufallszahlen(randMax, randAnzahl, liste);

     * Liste global setzen, damit sie auch weiterverwendet werden
     * kann und initialisieren;
     %global &liste.;
     %let &liste = ;

     data _null_;
```

```
          length zahlen $1000;
          * wir brauchen <randAnzahl> Werte;
          do i=1 to &randAnzahl.;
               * diese Variable gibt Auskunft, ob wir den Wert schon
               * in der Liste haben;
               neu=0;
               * neue Zahlen werden generiert, bis eine gezogen wird,
               * die noch nicht in der Liste ist;
               do until (neu eq 1);
                    * Erzeugung einer Zufallszahl zwischen 1 und
                    * <randMax>;
                    neuezahl = ceil(&randMax. * ranuni(0));
                    * Prüfung, ob Zahl schon vorhanden ist ;
                    if indexw(zahlen,neuezahl)=0 then neu=1;
                    else neu=0;
                    * put neuezahl= neu=;
               end;
               * nun kann die neue Zahl in die Liste aufgenommen
               * werden;
               zahlen = compbl(zahlen!!' '!!put(neuezahl,8.));
          end;
          call symput("&liste.",strip(zahlen));
     run;

%mend;
```

Dieses Makro wird aufgerufen, um die jeweils auszuwählenden Kredite zu bestimmen, die entweder aus der Kandidatenliste in das Portfolio übernommen werden, oder den umgekehrten Weg gehen sollen. Dies erfolgt in jeder Iteration auf folgende Art und Weise:

```
/* Anzahl der testweise neuen Investments ermitteln,            */
/* nach exponentieller diskreter Verteilung (Poisson)           */
%let anzrein = %sysfunc(ceil(%sysfunc(ranexp(0))));
/* Ermitteln der Zufallszahlen, deren Anzahl sich gerade        */
/* über ranexp ergeben hatte, diese stehen dann in listeIn      */
%zufallszahlen(&numNonPortf., &anzrein., listeIn);
/* Anzahl der Investments ermitteln, die testweise entfernt     */
/* werden sollen, nach exponentieller diskreter Verteilung      */
%let anzraus = %sysfunc(ceil(%sysfunc(ranexp(0))));
/* Ermitteln der Zufallszahlen, deren Anzahl sich gerade über   */
/* ranexp ergeben hatte, diese stehen dann in listeOut          */
%zufallszahlen(&numInPortf., &anzraus., listeOut);


/* Gesamte Investment-Tabelle durchlaufen, dabei gewählte       */
/* Zufallszahlen berücksichtigen                                */
data kandidatListe (drop=counter_in counter_out);
     set kandidatListe;
     /* Wenn bisher schon in Portfolio, entspr. Counter erhöhen */
     /* und prüfen, ob beibehalten oder verworfen wird.         */
```

```
     if inPortfolio=1 then do;
          counter_in + 1;
          /* Wenn die Counter-Zahl in der Liste unserer         */
          /* Zufallszahlen steht, dann raus.                    */
          if indexw("&listeOut.", counter_in)>0
               then inPortfolioTest=0;
               else inPortfolioTest=1;
          end;
          /* Wenn bisher nicht in Portfolio, entspr. Counter    */
          /* erhöhen                                            */
          /*und prüfen, ob jetzt rein oder nicht                */
     else do;
          counter_out + 1;
          /* Wenn die Counter-Zahl in der Liste unserer         */
          /* Zufallszahlen steht, dann rein.                    */
          if indexw("&listeIn.", counter_out)>0 then
               inPortfolioTest=1;
          else inPortfolioTest=0;
    end;
run;
```

Auf diese Weise ist es möglich, die Auswahl der zu wechselnden Kredite genau zu bestimmen.

3.3 Effiziente Implementierung mit zwei Tabellen

Allerdings waren wir auch mit dieser Implementierung des Algorithmus noch nicht ganz zufrieden, da es sich zeigt, dass es im Fall von einer besonders großen Zahl von zur Verbriefung zur Verfügung stehenden Krediten immer noch sehr lange dauert, bis wir ein annähernd optimales Portofolio ausgewählt haben. Dies rührt daher, dass wir in jeder Iteration zur Ermittlung der Fitness der neuen Konstellation stets durch die gesamte Liste gehen müssen. Der Zeitaufwand dieses Verfahrens steigt mit wachsender Liste sehr stark an. Aus diesem Grund sind wir wieder zurückgekehrt, zu der ursprünglichen Idee der zwei Tabellen. Allerdings behalten wir auf der einen Seite die lange Liste als Tabelle bei, in der sowohl die Kandidaten als auch die bereits für das Portfolio ausgewählten Kredite enthalten sind. Und auf der anderen Seite erzeugen wir in jeder Iteration, unter Berücksichtigung der durch die Zufallszahlen bestimmten lokalen Änderungen eine zweite Tabelle, die nur die für das Portfolio ausgewählten Kredite enthält. Die Fitness der neuen Lösung wird dann unter Verwendung dieser nur die ausgewählten Kredite enthaltenden Tabelle ermittelt:

```
/* Testliste erstellen, dabei gleich Investments raus         */
data testListe;
     set investListe;
     if indexw("&listeOut.", put(_n_,8.)) eq 0;
run;

/* Anzahl der testweise neuen Investments ermitteln,          */
/* nach exponentieller diskreter Verteilung (Poisson)         */
```

```
%let anzrein = %sysfunc(ceil(%sysfunc(ranexp(0))));


/* Neue Kandidaten in Testliste rein                              */
%let neu = 0;
%do %until (&neu. eq &anzrein.);

     /* Ermitteln einer Zufallszahl, diese stehen dann in der     */
     /* Macrovariablen testIn                                     */
     %zufallszahlen(&gesamtzahl., 1, testIn);

     /* Prüfen, ob Kontonummer schon in der Liste der             */
     /* Investments vorhanden ist.                                */
     proc sql noprint;
          select kontoNummer from testListe where kontoNummer in
          select kontoNummer from kandidatListe where
          numobs=&testIn.);
     quit;

     /* Wenn noch nicht vorhanden, kann die Kontonummer           */
     /* zugefügt werden.                                          */
     %if &SQLOBS. eq 0 %then %do;
          %let neu = %eval(&neu. + 1);
          proc sql noprint;
               insert into testListe select * from
               kandidatListe where numobs=&testIn.;
          quit;
     %end;
%end;
```

Wenn sich die Fitness aufgrund dieser neuen Auswahl verbessert hat, oder wenn der von der Temperatur abhängige Zufallsparameter das Beibehalten einer geringen Verschlechterung erlaubt, so übernehmen wir diese neue Auswahl und durchlaufen den Prozess aufs Neue:

```
/* Differenz der Fitness-Werte: bisheriger Wert minus            */
/* Herausforderer-Wert. Also ist der Wert positiv, wenn der      */
/* neue Wert besser als der alte ist.                            */
/* Je größer ein negativer Wert ist, umso schlechter der         */
/* Herausforderer.                                               */
%let fitnessDifferenz = %sysevalf(&fitnessBisher. -
&fitnessHerausforderer.);

/* Jetzt Wahrscheinlichkeit berechnen, ob Änderung trotz         */
/* Verschlechterung angenommen wird.  Dazu zunächst prüfen, ob   */
/* wir nicht schon auf minTemperatur abgekühlt haben und ohne    */
/* Annealing-Zusatz laufen ...                                   */
%if %sysevalf(&temp. gt &minTemp.) %then  %do;
     /* Zur Sicherheit nochmal den null-Fall abfangen,           */
     /* damit keine division by zero                             */
     /* auftritt. Ansonsten folgt hier die verwendete Gleichung */
```

```
     /* für das Simulated                                          */
     /* Annealing (e hoch Delta/Temperatur)                        */

     %if &temp. ne 0 %then
          %let prob =%sysevalf(%sysfunc(
               exp(1000*(&fitnessDifferenz.)/&temp.)));
     %else
          %let prob = 0;
          %let temp = %sysevalf(&temp. - &tempStep.);
     %end;
%else %let prob = 0;

%let trotzdem = 0;
%let zufall = %sysfunc(ranuni(-1));
%if %sysevalf(&zufall. lt &prob.) %then %let trotzdem = 1;


/* Wenn die Fitness sich verbessert hat, oder wenn Variable     */
/* "trotzdem" gleich 1 ist,                                     */
/* dann übernehmen wir die Werte.                               */
%if %sysevalf(&fitnessDifferenz. ge 0) or &trotzdem. eq 1
%then %do;

/* Wenn verbessert, dann neue Fitness und Investment-Anzahl     */
/* übernehmen                                                   */
%let fitnessBisher = &fitnessHerausforderer.;
%let numInPortf    = &numHerausforderer.;

/* Wenn verbessert, dann Liste der Investments im Portfolio     */
/* übernehmen                                                   */
proc datasets lib=work nolist;
     delete investListe;
     change testListe=investListe;
quit;

/* die Krit.- und Fitness-Werte in Gesamttabelle für            */
/* Fitness-Werte übertragen                                     */
proc append base = PfWerteGesamt_zweiV data = PfWerteAktuell;
quit;

/* Mit dieser Variablen merken wir uns die Iteration der        */
/* letzten Änderung.                                            */
%let lastChange = &i.;

%end;  /* Ende "Fitness tatsächlich verbessert" */
```

Dieser Ansatz führt zu einer sehr effizienten Implementierung des Simulated Annealing in SAS für große Kandidatenlisten. Das folgende Diagramm 1 zeigt einen Vergleich zwischen der Implementierung mit nur einer Tabelle gegenüber der Implementierung mit zwei Tabellen, also einer Portfolioliste und einer Gesamtliste, für den Fall eines kleinen Portfolios aus ca. 100 Datensätzen.

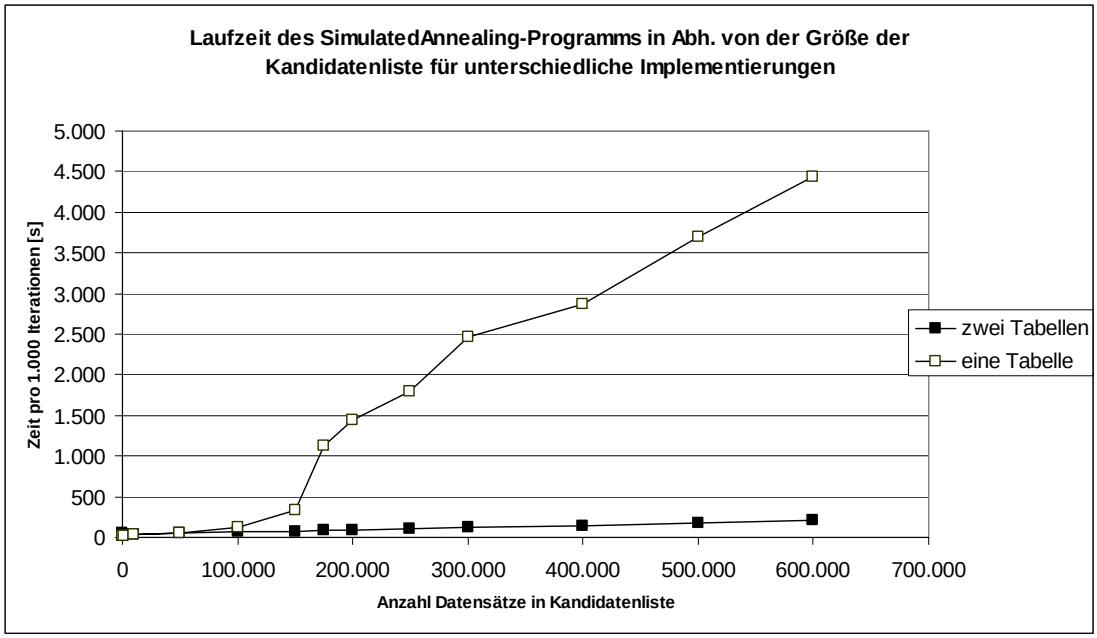

Diagramm 1: Laufzeit in Abhängigkeit von der Größe der Kandidatenliste bei einer Portfoliogröße von ca. 100 für unterschiedliche Implementierungen (schwarze Dreiecke: zwei Tabellen, graue Quadrate: eine Tabelle)

Für den Fall, dass das Portfolio einen höheren Anteil der Kandidaten umfasst, ist wiederum die Implementierung mit nur einer Tabelle günstiger. Dies illustriert Diagramm 2, in dem zusätzlich der Verlauf für Portfolien mit ca. 10.000 Datensätzen enthalten ist. Dabei ist die Implementierung mit nur einer Tabelle für alle betrachteten Fälle mit Kandidatenlisten von bis zu 100.000 Datensätzen stets die günstigere.

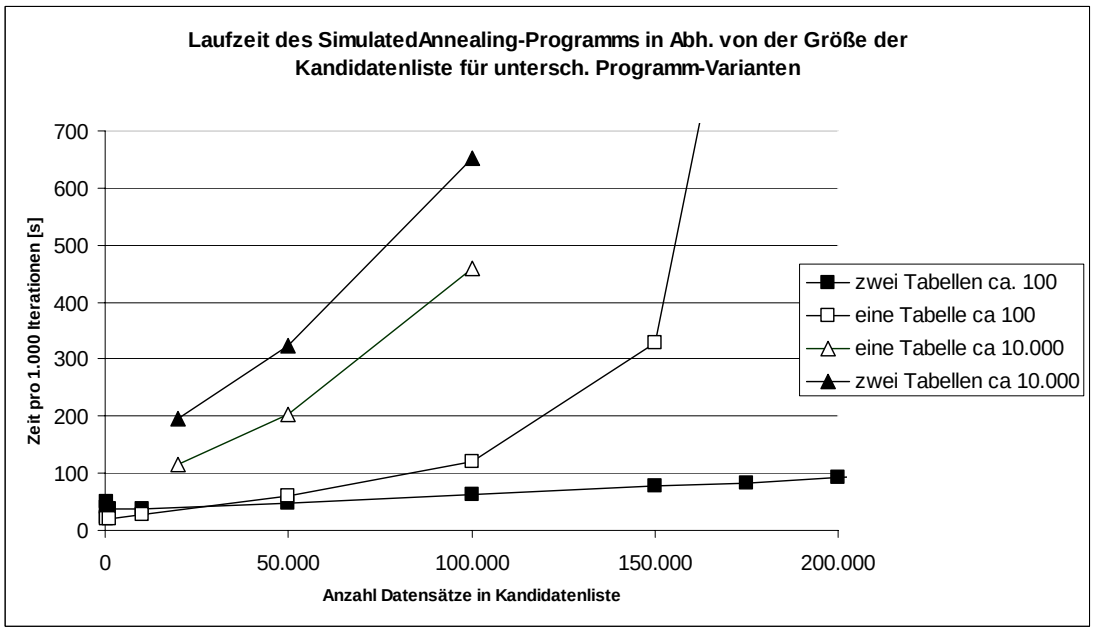

Diagramm 2: Laufzeit in Abhängigkeit von der Größe der Kandidatenliste für zwei Portfolios mit unterschiedlichen Größen (ca. 100 und ca. 10.000) und unterschiedlichen Implementierungen

In Tabelle 1 sind alle beobachteten Performance-Werte für die unterschiedlichen Implementierungen noch einmal aufgeführt.

Tabelle 1: Vergleich effizienter Implementierung mit einer und mit zwei Tabellen

		Zeit pro 1000 Iterationen [s]	
Anzahl in Portfolio	**Anzahl in Kandidatenliste**	**Eine Tabelle**	**Zwei Tabellen**
ca. 100	200	21,3	51,1
ca. 100	500	20,5	40,1
ca. 100	1.000	20,9	37,8
ca. 100	10.000	28,3	37,8
ca. 100	50.000	59,9	48,1
ca. 100	100.000	119,7	62,6
ca. 100	150.000	328,8	76,7
ca. 100	175.000	1.122,6	83,3
ca. 100	200.000	1.447,1	92,1
ca. 100	250.000	1.787,7	104,7
ca. 100	300.000	2.461,8	118,4
ca. 100	400.000	2.875,1	149,1
ca. 100	500.000	3.699,4	178,0
ca. 100	600.000	4.444,9	213,0
ca. 10.000	20.000	114,5	195,5
ca. 10.000	50.000	203,0	324,0
ca. 10.000	100.000	459,0	652,0

3 Ergebnisse des Simulated Annealing Prozess

Unabhängig von der Implementierungsmethode ergibt das Simulated Annealing vergleichbare Resultate. Im Folgenden werden für einen exemplarischen Fall der Verlauf der Fitness sowie eines weiteren Parameters für die aufeinanderfolgenden Iterationen dargestellt. Dabei wurden insgesamt 1000 Iterationen durchgeführt, wobei während der ersten 500 Iterationen die Temperatur abgesenkt wurde. Die Gesamtfitness verbessert sich und konvergiert mit fortschreitenden Iterationen gegen einen Optimalwert. Hierbei ist zu erkennen, dass anfangs, also bei erhöhter Temperatur, auch Fitnessverschlechterungen möglich sind, wie dies gemäß der Theorie des Simulated Annealing auch erwartet wird.
Dies gilt entsprechend auch für die einzelnen Kriterienwerte, sie nähern sich bei allen Implementierungsvarianten langfristig dem Optimalwert an, wie Diagramm 4 exemplarisch für einen Kriteriumswert zeigt. In diesem Fall wurde der Zielwert 0,75 vorgegeben.

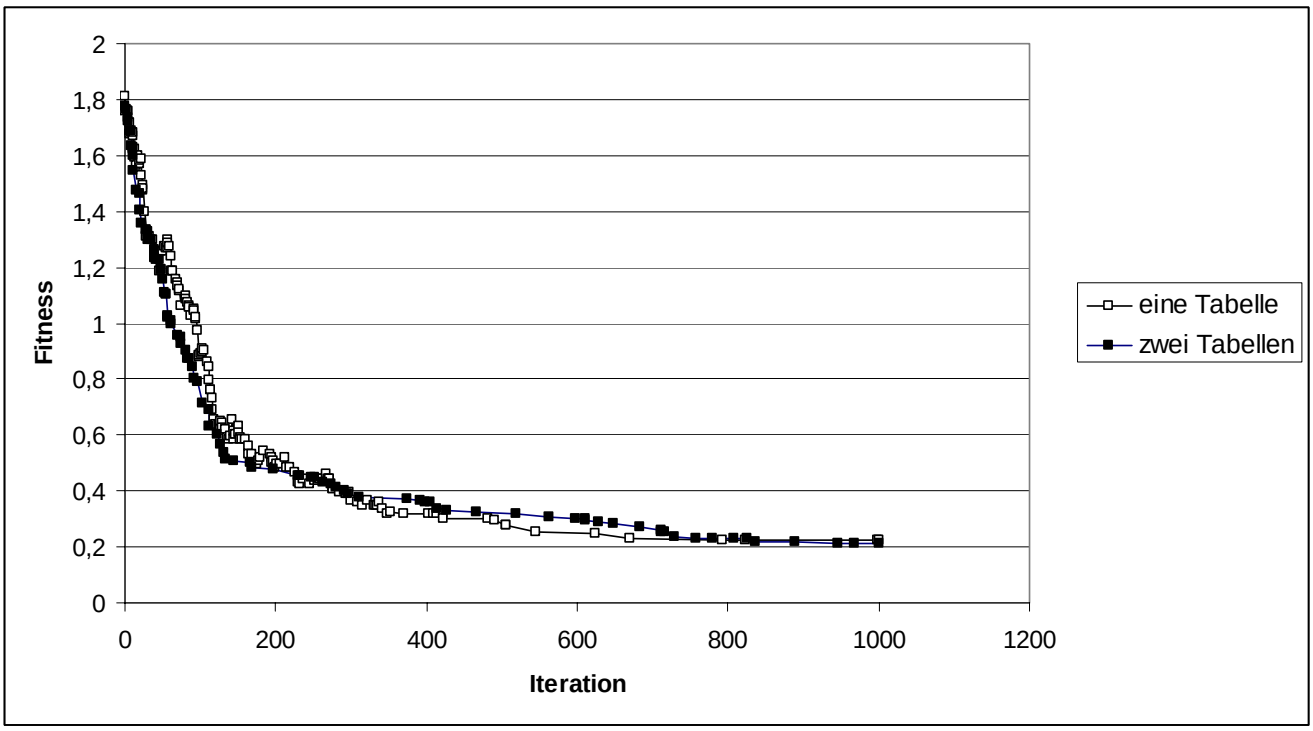

Diagramm 3: Verlauf der Fitnessfunktion für die Implementierung mit einer bzw. zwei Tabellen

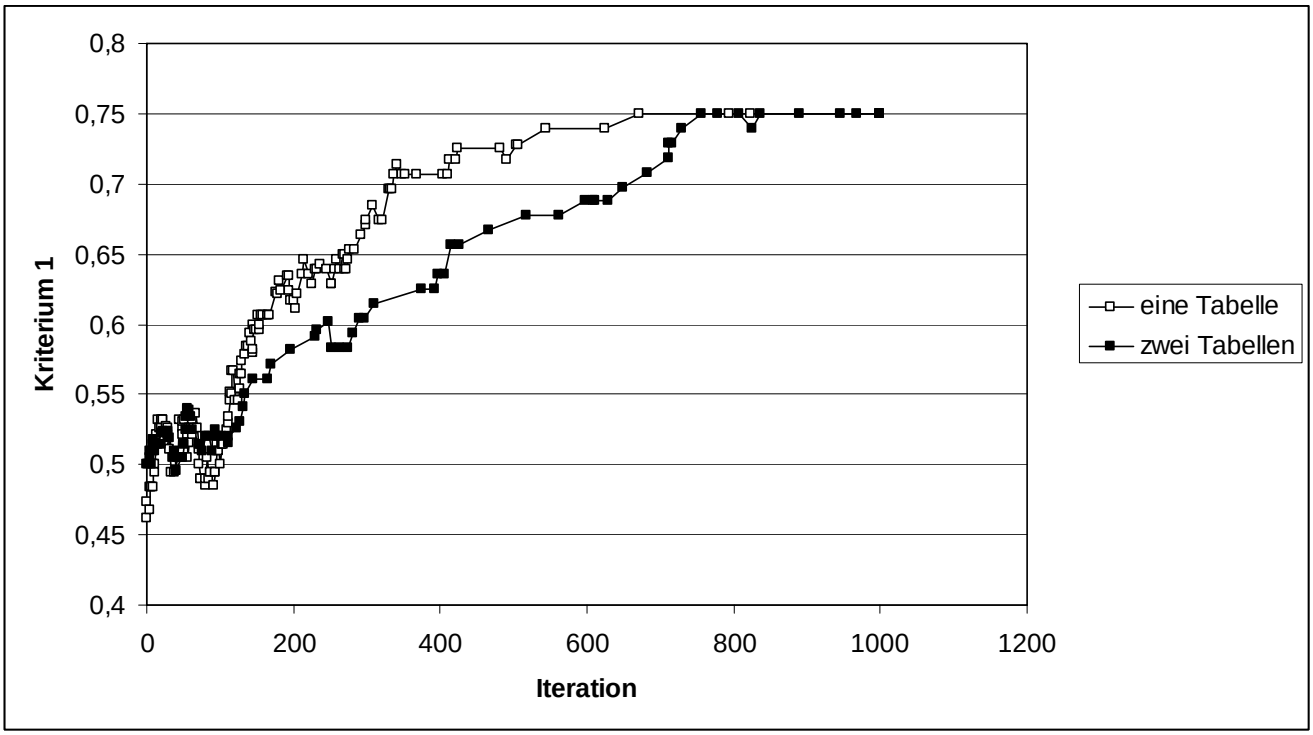

Diagramm 4: Änderung eines Kriteriumswerts im Verlauf des Simulated Annealing Prozesses für die Implementierung mit einer bzw. zwei Tabellen (Zielwert=0.75)

4 Ausblick

Die Implementierung von heuristischen Algorithmen in SAS ist zumindest nach unserer Erkenntnis in der Literatur noch nicht besonders eingehend behandelt worden. Aus diesem Grund halten wir es für angemessen, unsere in diesem Zusammenhang gemachten Erfahrungen hier darzustellen. Wir hoffen, damit eine Diskussion über die verschiedenen Möglichkeiten, solche Algorithmen in SAS zu implementieren, ins Leben zu rufen. SAS hat gegenüber anderen Programmierumgebungen einen großen Vorteil in Bezug auf die Geschwindigkeit, mit der große Datenmengen effizient verarbeitet werden können. Doch in vielen Situationen ist es auch wichtig, speziell programmierte Algorithmen möglichst effizient zu implementieren. Nach unserer Meinung ist es in vielen Situationen wichtig, diese beiden Eigenschaften zu kombinieren, um zufriedenstellende Ergebnisse zu erhalten. Dies ist uns mit dem in dieser Arbeit dargestellten Vorgehen auf jeden Fall gelungen.

Literatur

[1] T.J. Watsham, K. Parramore: Quantitative Methods in Finance. Thomson Business Press, London, 1997.

[2] Z. Michalewicz, D.B. Fogel: How to Solve It: Modern Heuristics. Springer, Heidelberg, 2000.