Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Studienarbeit SA-2004-04

Evolutionäre Optimierung von Synchronisationsalgorithmen

Christian Hitz, chitz@ee.ethz.ch

Stefan Schuler, stschule@ee.ethz.ch

Betreuer: Philipp Blum, blum@tik.ee.ethz.ch

Verantwortlicher: Prof. Dr. Lothar Thiele

Wintersemester 2003/2004

20. Februar 2004

Inhaltsverzeichnis

1 Einleitung **4**
1.1 Hintergrund 4
1.2 Übersicht über die Arbeit 5

2 Tracefiles **6**
2.1 Warum wir Tracefiles verwenden 6
2.2 Inhalt eines Tracefiles 6
2.3 Szenarien 7
2.4 Reale Lastgenerierung mit Streaming 7
2.5 Aufnahme 7
2.6 Analyse 9

3 Simulation mit PISA **14**
3.1 Konzept 14
3.2 Implementation 15
3.2.1 Mutation / Rekombination 15
3.2.2 Algorithmen 15
3.2.3 Bewertungskriterien 16
3.2.4 Verwendete Kriterien 18

4 Optimierung mit PISA **19**
4.1 Tools 19
4.1.1 3D-Darstellung der Paretopunkte 19
4.1.2 Verschiebung der Paretofront über die Generationen . 19
4.1.3 Parameter 19
4.2 Resultate 22
4.2.1 Algorithmen 22
4.2.2 Synchronisationsintervalle 22
4.2.3 Szenarien 23
4.2.4 Guter MTIE auf Kosten hoher Setup Zeit 24
4.2.5 Kriterien 24

5 Schlussfolgerungen **26**

A Bedienung der PISA Tools **27**
A.1 Simulationsablauf 27
A.2 Format des Parameterfiles 28
A.3 Einfügen eines neuen Algorithmus 29
A.4 Einfügen eines neuen Kriteriums 29

B Projektbeschrieb **31**
B.1 Einleitung 31
B.2 Aufgabenstellung 32
B.3 Durchführung der Semesterarbeit 33
B.3.1 Allgemeines 33
B.3.2 Material 33
B.3.3 Abgabe 34

C Zeitplan **35**

D Files auf CD **37**

E C Source Code **39**
E.1 variator_user.c 39
E.2 variator_user.h 66
E.3 evaluate_parameter.c 71
E.4 evaluate_parameter.h 75
E.5 variator_param.txt 75

F MATLAB Skripte **77**
F.1 analyse_parameter.m 77
F.2 dominatePts.c 78
F.3 paretoplot3d.m 80
F.4 pareto_movie.m 81
F.5 pareto_plot.m 81
F.6 parsemain.m 83
F.7 analyse_trace.m 83

Kapitel 1

Einleitung

1.1 Hintergrund

Mit der weiten Verbreitung von Wireless Netzwerken (IEEE 802.11b) wird diese Technologie auch für verteilte Audioanwendungen (z.B. drahtlose Lautsprecher) interessant. Für diese Audioanwendungen darf sich eine gemeinsame Zeit auf allen Netzwerkknoten nicht mehr als $10\,\mu$s unterscheiden, da die Differenz sonst hörbar wird. Mit Zeitnachrichten wird die Zeit im Netzwerk bekannt gemacht. Die Übertragungsbedingungen wie Nachrichtendelays und Nachrichtenverluste sind in Wireless Netzwerken jedoch nicht konstant und werden stark von der Netzwerktopologie und der Umgebung beeinflusst. Wird die Zeit aus einer Zeitnachricht übernommen, liefert dies eine zu ungenaue Synchronisation. Um trotzdem eine gemeinsame Zeit zu erhalten, werden Zeitsynchronistationsalgorithmen (Clock Synchronization Algorithms: CSA) eingesetzt.

Ein CSA operiert auf den empfangenen Zeitnachrichten und auf den lokalen Empfangszeiten der jeweiligen Zeitnachrichten. Daraus berechnet er dann eine Schätzung des lokalen Uhrendrifts und eine synchronisierte Zeit. Es gibt verschiedenste CSAs, unter anderem auch LSDC (beschrieben in [6]). Der LSDC Algorithmus hat sieben Parameter, die alle die Qualität der erreichbaren Synchronisation beeinflussen. Eine manuelle Einstellung der Parameter ist sehr schwierig. Die Idee unserer Arbeit war es nun, diese Parameter mit evolutionären Methoden zu optimieren. Dazu verwendeten wir das am TIK entwickelte "Platform and Programming Language Independent Interface for Search Algorithms" kurz PISA [1].

PISA bietet eine Schnittstelle zwischen einem Optimierungsproblem und einem Suchalgorithmus. Dadurch konnten wir unsere Optimierungen durchführen, ohne uns näher mit Suchalgorithmen beschäftigen zu müssen.

1.2 Übersicht über die Arbeit

In Kapitel 2 beschreiben wir, wie wir die Tracefiles für die Simulation aufgenommen haben, wie wir die Last mit einer realen Streaming-Applikation generiert haben und wir präsentieren Analysen der Tracefiles. Die Konzepte und Implementation des Tools mit PISA sind in Kapitel 3 beschrieben. Die Auswertung der PISA Simulation und die erzielten Resultate werden in Kapitel 4 diskutiert. Nach der Schlussfolgerung im Kapitel 5 folgt im Anhang A eine Übersicht über die Bedienung der PISA Programme. Ebenfalls dort beschreiben wir, wie man neue Algorithmen und Kriterien in unser Programm einfügen kann.

Kapitel 2

Tracefiles

In diesem Kapitel geht es um die Aufnahme von neuen Tracefiles. Wir untersuchen die Delays auf einem 802.11b Wireless LAN Netzwerk in verschiedenen Szenarien, bei unterschiedlichen Laststärken und Synchronisationsintervallen. Die Last wird dabei von einer realen Streaming Applikation generiert.

2.1 Warum wir Tracefiles verwenden

Unsere Simulation arbeitet nicht mit wirklichen Zeitnachrichten, die während der Simulation ausgetauscht werden, sondern mit einem Tracefile, in welchem vorher aufgezeichnete Zeitstempel abgespeichert sind. Damit ist die Wiederholbarkeit und Vergleichbarkeit gewährleistet. Trotzdem basieren die Daten auf realistischen Aufnahmen.

2.2 Inhalt eines Tracefiles

Für eine genaue Beschreibung der Tracefiles und der Delaymessung verweisen wir auf [4]. Hier soll nur kurz auf den Inhalt eingegangen werden.

- In den "*ref.txt" Files werden zum gleichen Zeitpunkt die lokalen Zeiten von Sender und Empfänger ins Logfile geschrieben. Damit erhalten wir eine globale Referenzzeit. Das Format ist:
  `cnt     sendtime [s us]     receivetime [s us]`
- In den "*sync.txt" Files werden die lokalen Sende- und Empfangszeiten ins Logfile geschrieben, auch im Format:
  `cnt     sendtime [s us]     receivetime [s us]`

Aus diesen Angaben lässt sich der Delay wie folgt berechnen:

Delay = Referenzzeit beim Empfang – Referenzzeit beim Senden

Mit dem MATLAB Skript `parsemain` berechnen wir aus den ref und sync Files ein neues File mit normalisierten Zeiten.

- Das Format der normalisierten Tracefiles ist:
  `local_time     receive_time     reference_time`
  `local_time` ist die lokale Zeit des Senders beim Abschicken der Zeitnachricht. `receive_time` ist die lokale Zeit des Empfängers beim Empfang. `reference_time` ist der Zeitpunkt des Sendens nach der Uhr des Empfängers.

2.3 Szenarien

Wir haben all unsere Experimente in einem 802.11b Wireless LAN durchgeführt, weil in Wireless Netzwerken der Delay grosse Schwankungen aufweist, was hohe Anforderungen an den Synchronisationsalgorithmus stellt. Wir haben folgende Szenarien getestet (Abbildung 2.1):

- **Ad-Hoc Modus:** Zeitnachrichten und die Streaming Pakete können direkt zwischen den Linux PCs über die Wireless Karte ausgetauscht werden.
  - Last und Zeitnachrichten haben die gleiche Richtung
  - Last und Zeitnachrichten haben verschiedene Richtungen
- **Infrastructure Modus:** Ein Linux PC dient als Server und broadcastet seine Zeitnachrichten und Streaming Pakete über den Accesspoint (AP), mit dem er mit einem direkten Kabel verbunden ist, an die Empfänger.

2.4 Reale Lastgenerierung mit Streaming

Bis anhin wurde die Last künstlich generiert. Um eine wirklichkeitsnahe Lastverteilung zu erreichen, generieren wir die Last indem wir mit dem Video Lan Client (VLC) [2] Musik oder Video per UDP über das Netzwerk streamen.

Die kleine Last generieren wir mit einem MP3 Stream (128 KBit/s). Er hat eine konstante Übertragungsrate. Die mittlere und grosse Last erzeugen wir mit DivX Video Streams, welche eine variable Übertragungsrate haben. Wie in Abbildung 2.2 zu sehen ist, schwankt die Rate beim 3 MBit/s Video zwischen 100 und 700 KByte/s, je nachdem, ob eine ruhige oder bewegte Szene übertragen wird.

2.5 Aufnahme

Die Aufnahme der Tracefiles haben wir wie in [4] beschrieben durchgeführt. Dort wurden Traces bei verschiedener Last in wireless und wired Netzwerken

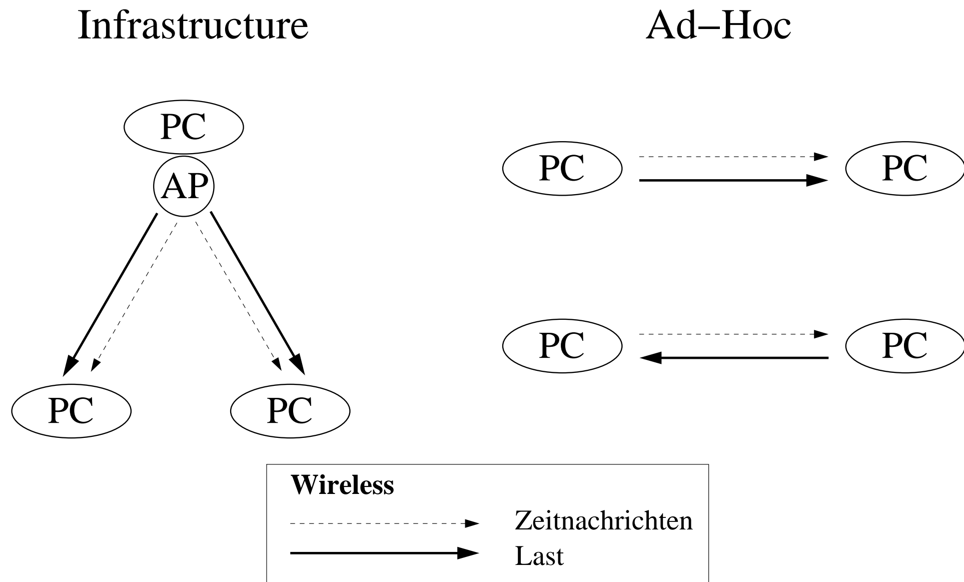

Abbildung 2.1: Infrastructure und Ad-Hoc Szenarien

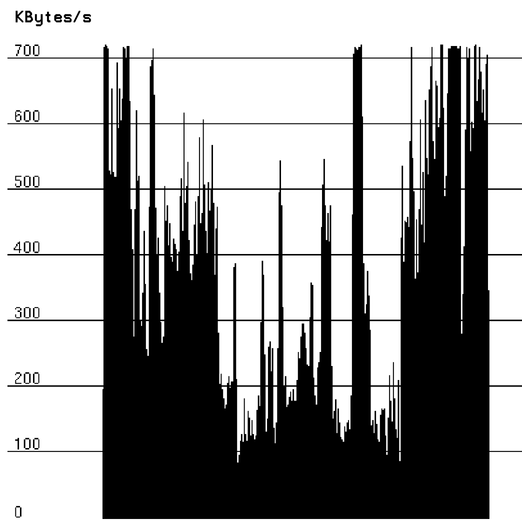

Abbildung 2.2: Aufzeichnung der Last (3Mbit/s) über 10 Minuten

aufgenommen. Wir haben zusätzlich die Synchronisationsintervalle variiert und zwecks Mittelung je 10 Durchläufe gemacht.

Folgende Parameter haben wir miteinander kombiniert:

- Durchläufe: [1] - [10]
- Szenario: Infrastructure (In), Ad-Hoc gleiche Richtung (AdGr), Ad-Hoc verschiedene Richtungen (AdVr)
- Last: keine, 128 KBit/s (klein), 1 MBit/s (mittel), 3 MBit/s (gross)
- Synchronisations Intervall: 20 ms, 200 ms, 2000 ms

Beispiel Tracefile

[01]AdGrklein200.txt:
1. Durchlauf, Ad-Hoc gleiche Richtung, kleine Last, 200 ms Synchronisationsintervall.

Insgesamt gibt das 10*3*4*3 = 360 Kombinationen. Damit wir nicht 360 Mal die Lastgenerierung, den Sender und Empfänger der Zeitnachrichten mit unterschiedlichen Einstellungen von Hand starten und stoppen müssen, haben wir die beiden Shell Skripte `runTracesAdHoc.sh` und `runTraces Infrastructure.sh` geschrieben. Sie führen für das jeweilige Szenario einen Durchlauf automatisch aus.

2.6 Analyse

In diesem Abschnitt besprechen wir den Einfluss der Laststärke und der Synchornisationsintervalle auf die Verteilung (PDF/CDF) der Delays. Zudem untersuchen wir, in welchen Abständen Zeitnachrichten beim Empfänger ankommen.

Last

Als erstes wollen wir uns die Verteilung der Delays bei unterschiedlichen Lasten anschauen. Dazu vergleichen wir die Probabilistic Density Function (PDF) mit der Cumulated Density Function (CDF) über 10 Tracefiles gemittelt. In den Abbildungen 2.3 - 2.5 sind die Vergleiche für verschiedene Szenarien dargestellt. Der minimale Delay ist unabhängig von der Last. Bei den beiden Ad-Hoc Szenarien liegt er etwa bei 835 μs, beim Infrastructure bei 1020 μs. Bei grösserer Last hat es mehr Pakete mit einem hohen Delay.

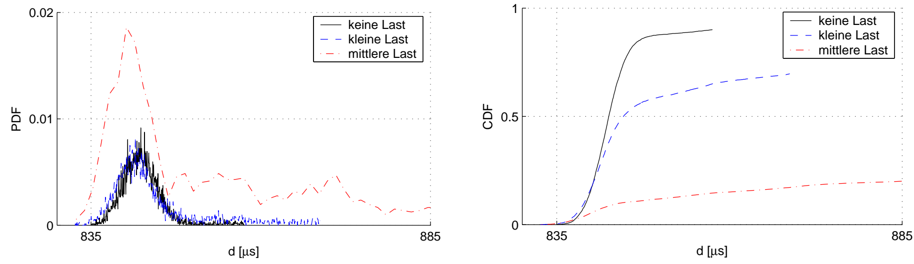

Abbildung 2.3: Vergleich der PDF und CDF bei verschiedenen Lasten. Szenario: Ad-Hoc verschiedene Richtungen, 200 ms

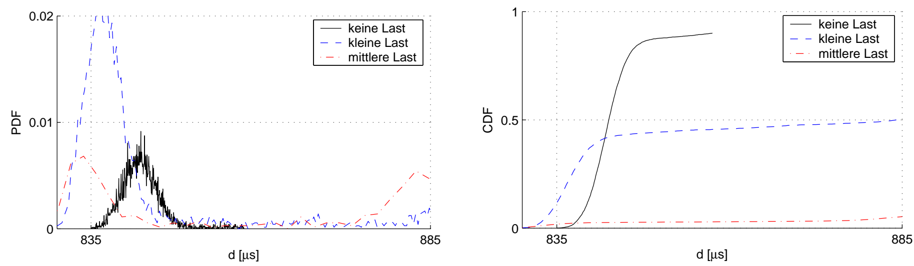

Abbildung 2.4: Vergleich der PDF und CDF bei verschiedenen Lasten. Szenario: Ad-Hoc gleiche Richtung, 200 ms

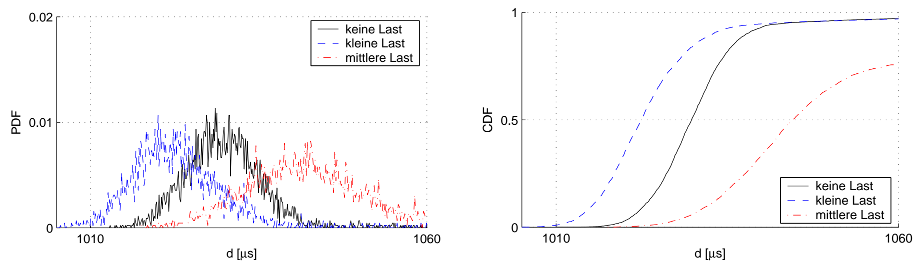

Abbildung 2.5: Vergleich der PDF und CDF bei verschiedenen Lasten. Szenario: Infrastructure, 200 ms

Synchronisationsintervalle

Die Abbildungen 2.6 bis 2.8 vergleichen die PDF und CDF Verteilung gemittelt über 10 Tracefiles für unterschiedliche Synchronisationsintervalle.

Bei allen Lasten ändert sich die Verteilung nicht wesentlich. Die Synchronisationsintervalle haben somit keinen Einfluss auf die Verteilung der Delays.

Ein kleines Detail sei hier noch erwähnt: Bei keiner Last (Abbildung 2.6) hat die Kurve für 20 ms einen kleineren Delay als diejenige für 200 und 2000 ms. Bei mittlerer Last (Abbildung 2.8) ist es genau umgekehrt. Offenbar verursacht hier die Last, die durch mehr Zeitnachrichten entsteht, Kollisionen mit der Streaming Last. Das wirkt sich negativ auf den Delay aus.

Abstände der Zeitnachrichten

Uns interessierte auch in welchen Abständen die Zeitnachrichten tatsächlich beim Empfänger ankommen. Dies haben wir in Abbildung 2.9 dargestellt.

Die drei Linien um 30 ms rühren daher, dass der Sender der Zeitnachrichten nicht genau alle 20 ms ein Paket schickt, sondern auf das nächste Time Slice von 10 ms warten muss (C-Funktion `usleep`) bis das Paket geschickt wird. Die Punkte um 60 ms bedeuten, dass eine Zeitnachricht verloren ging. Das kann vorkommen, weil wir die Zeitnachrichten im Broadcast Modus übertragen.

Bei grosser Last sieht man, dass der Abstand der Zeitnachrichten zufälliger verteilt ist. Jetzt variiert der Delay auf dem Shared Medium stark.

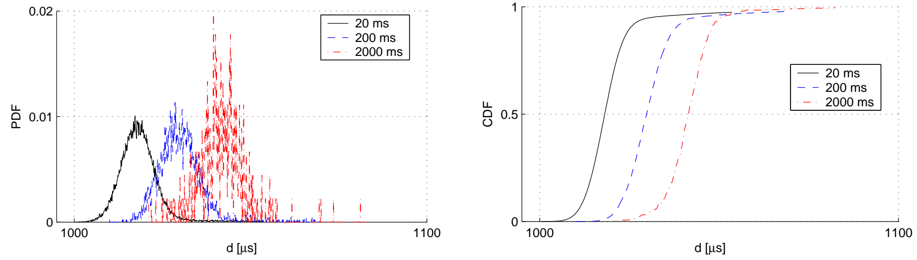

Abbildung 2.6: Vergleich der PDF und CDF bei verschiedenen Synchronisationsintervallen. Szenario: Infrastructure, keine Last.

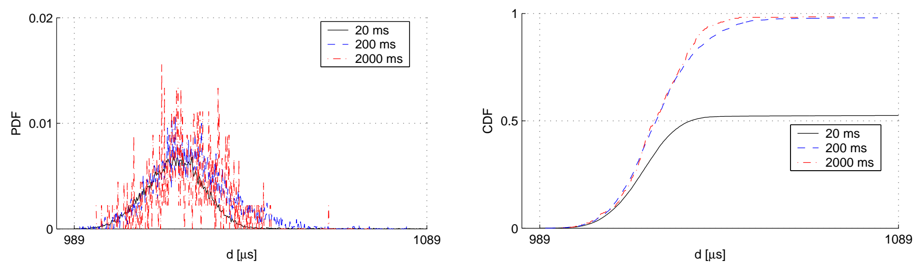

Abbildung 2.7: Vergleich der PDF und CDF bei verschiedenen Synchronisationsintervallen. Szenario: Infrastructure, kleine Last.

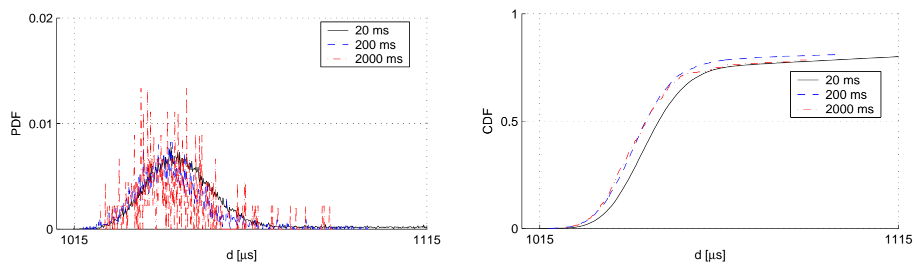

Abbildung 2.8: Vergleich der PDF und CDF bei verschiedenen Synchronisationsintervallen. Szenario: Infrastructure, mittlere Last.

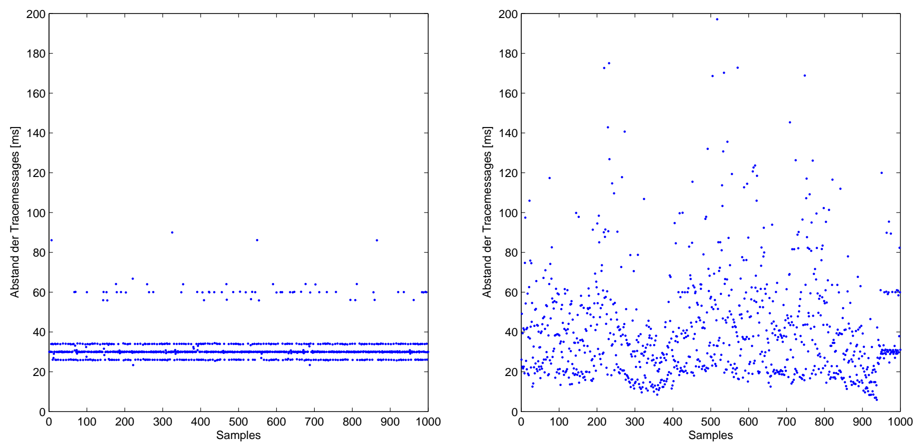

Abbildung 2.9: Abstand der Zeitnachrichten beim Empfänger bei kleiner Last (links) und bei grosser Last (rechts) im Infrastructure Modus

Kapitel 3

Simulation mit PISA

In diesem Kapitel stellen wir die unterschiedlichen Algorithmen und Bewertungskriterien vor, die wir in Form eines Variators für PISA implementiert haben. Auch erläutern wir die verwendeten Methoden zur Mutation / Rekombination für die evolutionäre Optimierung.

3.1 Konzept

Um die Optimierung mit PISA durchzuführen, gibt es zwei Möglichkeiten: erstens die Integration von PISA mit der bereits bestehenden MATLAB Simulation oder zweitens die Portierung des Algorithmus nach C. Wir haben uns für die zweite Möglichkeit entschieden, da wir so auf die Hilfsfunktionen von PISAlib zurückgreifen können.

In PISA wird mit Individuen gearbeitet. Ein Individuum besteht aus seinen Genen und aus Zielwerten. Während des Prozesses der Optimierung werden die Zielwerte minimiert.

In unserem Fall sind die Gene des Algorithmus seine Parameter. Die Zielwerte sind Kennwerte wie: Jitter, MTIE (Maximum Time Interval Error), usw.

Die Parameter werden bei der Optimierung mutiert und rekombiniert. Für die Rekombination eines Parameters zwischen zwei Individuen sehen wir verschiedene Möglichkeiten, die unter Umständen auch alle berücksichtigt werden können:

- Vertauschen von zwei Parametern
- Mittelwert
- Gewichteter Mittelwert (Neuer Wert liegt näher beim Wert des stärkeren Vorfahren)

Für die Mutation ist eine prozentuale Änderung in eine zufällige Richtung denkbar.

3.2 Implementation

Die erwähnten Funktionen in den nächsten Abschnitten sind im Anhang E abgedruckt.

3.2.1 Mutation / Rekombination

Die Mutation / Rekombination ist in der Funktion `evaluate()` implementiert. Für jedes neu zu erzeugende Individuum (Child) werden zufällig zwei Parents ausgewählt. Das Child wird als Kopie von Parent1 initialisiert. Mit Wahrscheinlichkeit 0.5 wird nun ein zufälliger Parameter im Bereich ±`MUTATION_FACTOR` verändert (siehe A.2). In den anderen Fällen wird rekombiniert. Dabei werden eine zufällige Anzahl von Parametern von Parent2 in das Child kopiert.

3.2.2 Algorithmen

Die Algorithmen werden in der Funktion `calculate_ov()` aufgerufen. Als Argumente erhalten sie das Parameter-Array, die "local time", und die "remote time". Als Rückgabe liefern sie mindestens die synchronisierte Zeit t_s und den lokalen Uhrendrift. Folgende Algorithmen sind implementiert:

Local Selection with Drift Compensation (LSDC6): Der LSDC Algorithmus wird in [6] genau beschrieben. Er hat 7 Parameter:

- DRIFT_VAR_INIT: Initialer Wert für die Driftkompensation.
- DRIFT_VAR_FACTOR: Wert der Veränderung der Driftkompensation.
- DRIFT_VAR_FINAL: Finaler Wert der Driftkompensation.
- INIT_PHASE: Anzahl Zeitstempel bis die erste Driftkompensation vorgenommen wird.
- WINDOW_SIZE: Grösse der FIFO-Queue.
- ESTIM_ERROR: Geschätzter Fehler.
- DRIFT_MAX: Maximaler Drift.

Phased Local Loop (PLL2): Algorithmus basierend auf einem PI-Regler. Die Parameter sind die Proportionalkonstante κ_I, die Integratorkonstante κ_P und die maximale Differenz zwischen erhaltenem Zeitstempel und synchronisierter Zeit θ_{max}.

Continuous Linear Least Square Regression (LLR_CONT): LLR_CONT berechnet die lineare Regression in jedem Schritt über die letzten L Zeitnachrichten. L ist der einzige Parameter.

MPEG_GRAD ist im wesentlichen eine speicheroptimierte LLR. Es muss immer nur ein Durchschnitt gespeichert werden, statt der L letzten Zeitstempel. Der Parameter ι gibt an, wie stark der neu berechnete Wert gegenüber der bisherigen Kompensation gewichtet wird.

In Sektion A.3 wird beschieben, wie ein neuer Algorithmus eingefügt werden kann.

3.2.3 Bewertungskriterien

Jitter

Der Jitter wird in `jitter()` oder `jitterslow()` berechnet. Die Definition des Jitters ist wie folgt:

$$\text{Jitter}(\tau, T) = \max_{\tau \le t \le T} (\text{TE}(t)) - \min_{\tau \le t \le T} (\text{TE}(t))$$

wobei τ der Auswertungszeitpunkt und T die Messperiode darstellen. $\text{TE}(t)$ ist der Fehlervektor berechnet als $t_s - t_{ref}$.

Wir werten den Jitter an der Stelle $\tau =$ `EVAL_TIME` aus. Die Implementation in `jitter()` beginnt mit der Berechnung am Ende des Errorvektors und muss dadurch nur einmal das Array durchgehen. Die Komplexität ist somit $\mathcal{O}(n)$ statt $\mathcal{O}(n^2)$ wie in `jitterslow()`.

Daneben berechnet `jitter()` auch noch die Setup Time für den Jitter. Die Setup Time ist die Zeit, zu der der Jitter mehr als 10% grösser ist als bei `EVAL_TIME` (analog zu MTIE in Abbildung 3.2).

Maximum Time Interval Error (MTIE)

Der MTIE ist formal wie folgt definiert, wobei $\text{TE}(t)$ für den Fehlervektor, wie oben definiert, steht:

$$\text{MTIE}(\tau, T) = \max_{0 \le t_0 \le T - \tau} \left\{ \max_{t_0 \le t \le t_0 + \tau} (\text{TE}(t)) - \min_{t_0 \le t \le t_0 + \tau} (\text{TE}(t)) \right\}$$

Der MTIE ist zwei Mal implementiert als `mtieslow()` und als `mtie()`. Durch eine Beschränkung der Fenstergrösse τ auf 2^n Zeitstempel kann die Berechnung mit binärer Dekomposition implementiert werden (siehe [7]). Dadurch verbessert sich die Komplexität der Berechnung von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n \log n)$.

Die Setup Time des MTIE ist definiert als der Zeitpunkt, an dem der MTIE mehr als 10% von der Differenz zwischen MTIE am Anfang und MTIE bei 1/3 der Samples abweicht.

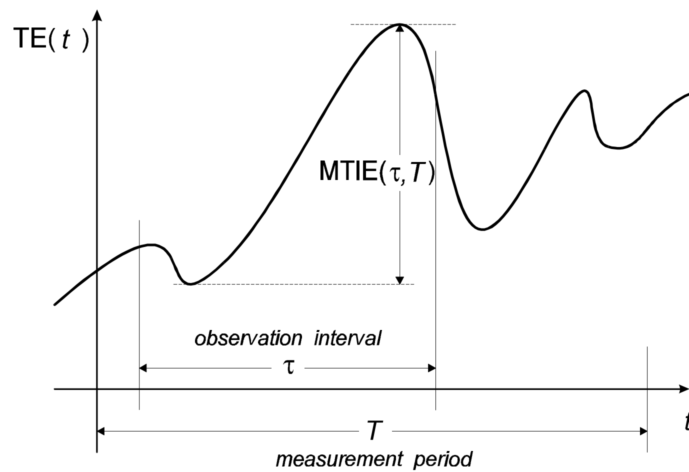

Abbildung 3.1: Definition des MTIE (Quelle: [7])

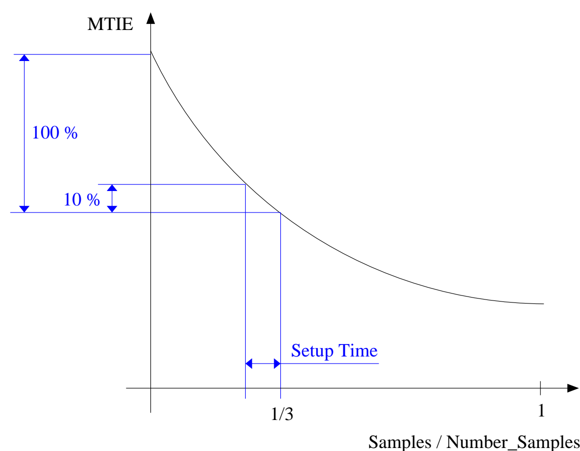

Abbildung 3.2: Definition der Setup Zeit

Time Variance (TVAR)

Auch die TVAR haben wir einmal langsam (`tvarslow()`) und einmal schnell (`tvar()`) implementiert. Die schnelle Version benutzt die Rekursionsformel aus [7].

Settling Time

Die Settling Time ist die Dauer, bis sich der Error auf unter $\pm 10\%$ eingeschwungen hat. Implementiert in `settling_time()`.

Der Nachteil der Settling Time gegenüber der Setup Time ist, dass sie von der absoluten Grösse des durchschnittlichen Errors abhängig ist.

Verschlechterung

Als Verschlechterung haben wir den Quotienten

$$\frac{\emptyset(\text{MTIE bei grosser Last})}{\emptyset(\text{MTIE bei kleiner Last})}$$

definiert.

3.2.4 Verwendete Kriterien

Bei unseren Optimierungen haben wir meistens mit den folgenden drei Zielwerten gerechnet:

1. Durchschnitt des MTIE von 10 Tracefiles mit kleiner Last
2. Durchschnitt des MTIE von 10 Tracefiles mit grosser Last
3. Durchschnitt der Setup Time des MTIE von 10 Tracefiles mit kleiner Last

Diese Kombination hat uns in der Simulation gute Konvergenz geliefert.

Statt des 2. Kriteriums von oben haben wir auch mit der Verschlechterung optimiert. Der Einbezug der Verschlechterung ist aber nicht ideal, da sie genau im Gegensatz zum Kriterium 1 steht. D.h. gute Verschlechterung = schlechter Durchschnitt des MTIE bei kleiner Last.

Kapitel 4

Optimierung mit PISA

In diesem Kapitel werden wir zuerst die MATLAB-Tools zur Auswertung der Simulationen vorstellen. Im zweiten Teil werden wir dann die Resultate präsentieren, die wir mit der Simulation erreicht haben. Wir vergleichen verschiede Algorithmen untereinander, untersuchen unterschiedliche Synchronisationsintervalle und Wireless-Szenarien und analysieren den Effekt von verschiedenen Bewertungskriterien.

4.1 Tools

4.1.1 3D-Darstellung der Paretopunkte

Mit `paretoplot3d` können wir die Paretopunkte einer Generation dreidimensional darstellen. Zusätzlich werden auch die drei Projektionen auf die Koordinatenebenen gezeichnet. Jeder Punkt in Abbildung 4.1 entspricht einem Individuum mit seinem Parametersatz.

4.1.2 Verschiebung der Paretofront über die Generationen

Um uns eine Vorstellung zu vermitteln wie eine Simulation abläuft, stellen wir mit `pareto_movie` jede Generation als Frame in einem Film dar. Wir können so beobachten, wie sich die Projektion der 3-dimensionalen Paretofront während der Simulation verschiebt.

Da es in MATLAB sehr lange dauern kann bis die Paretopunkte gefunden sind, haben wir die Funktion, welche die Paretopunkte bestimmt, in C implementiert. Wir können sie dadurch 7 Mal schneller finden.

4.1.3 Parameter

Mit `analyse_parameter` kann man die Auswirkungen der Parameter auf die Bewertungskriterien sichtbar machen.

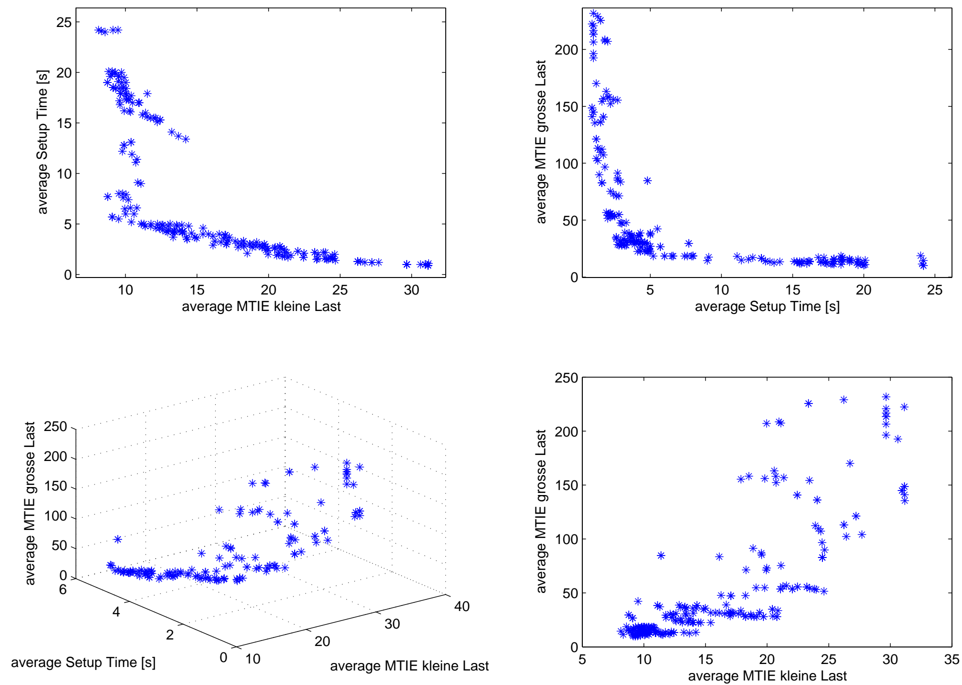

Abbildung 4.1: unten links: 3D Ansicht der Paretopunkte einer Generation; oben und rechts: alle Projektionen der 3D-Paretopunkte.

Auf der X-Achse der Abbildung 4.2 sind die Individuen aufgetragen. Die oberen drei Plots beinhalten die Bewertungskriterien, darunter sind die entsprechenden Parameter. Jetzt lassen sich Vergleiche anstellen wie:

- Im Bereich von Individuum 70 bis 120 ändert nur der `ESTIM_ERROR`. Daraus lässt sich schliessen, dass er nicht so relevant ist.

- Im Bereich von Individuum 150 bis 200 ändern nur die `INIT_PHASE` und die `Setup_Time`. Daraus lässt sich ableiten, dass die beiden eng miteinander korreliert sind.

- Nach Individuum 240 ist ein starker Anstieg beim `DRIFT_VAR_FINAL` zu erkennen. Gleichzeitig steigen auch `MTIE_klein` und `MTIE_gross` an. Diese drei Grössen scheinen also auch zusammenzuhängen.

Dabei ist anzumerken, dass diese Aussagen nur brauchbar sind, wenn die Parameter etwa in dem Bereich sind, in dem die Beobachtung gemacht wurde. Man kann sie nicht einfach verallgemeinern.

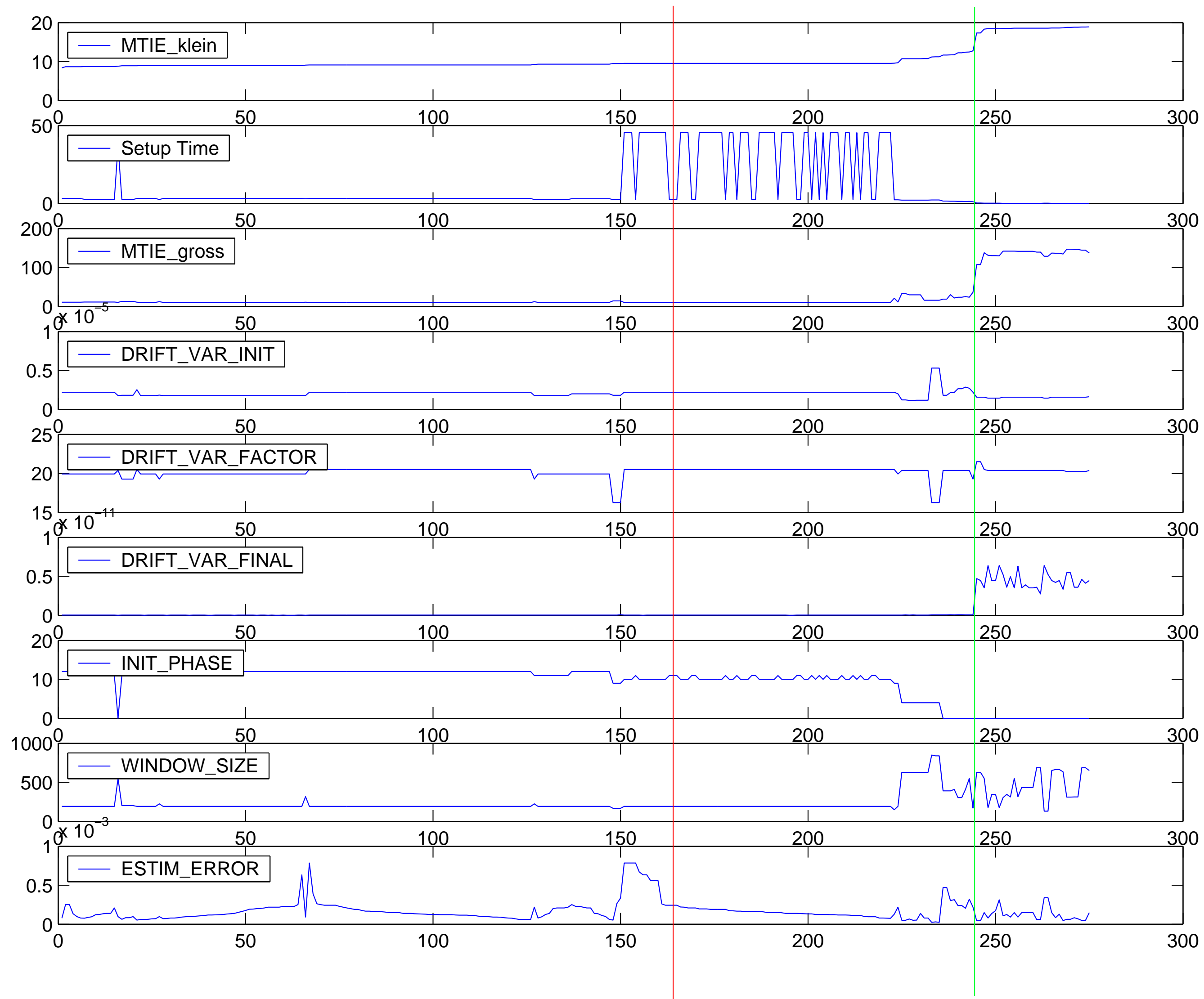

Abbildung 4.2: Parameter Darstellung

4.2 Resultate

4.2.1 Algorithmen

Ein Vergleich unter den vier implementierten Algorithmen hat einige Unterschiede aufgezeigt. Der LSDC6 erzielt in allen drei Kriterien gute Werte. Der PLL2 erreicht die gleichen MTIE-Werte wie der LSDC6, hat aber eine wesentlich höhere Setup Zeit (siehe Abbildung 4.3). LLR und GRADIENT erreichen zwar gute Setup Zeiten und gute MTIE's bei kleiner Last, sind aber miserabel bei grosser Last (siehe Abbildung 4.4). Zusammengefasst resultieren die folgenden typischen Werte:

	MTIE kleine Last [μs]	MTIE grosse Last [μs]	Setup Time [s]
LSDC6	8	10	5
PLL2	9	10	60
LLR / GRADIENT	12	2500	17

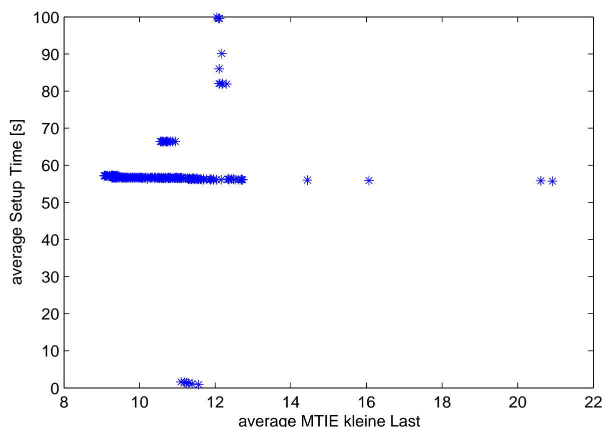

Abbildung 4.3: Zeigt die sehr hohe Setup Zeit in Relation zum MTIE bei kleiner Last beim PLL2 Algorithmus.

4.2.2 Synchronisationsintervalle

Bei den 3 Synchronisationsintervallen (20 ms, 200 ms, 2000 ms) konnten wir keine signifikanten Unterschiede bei den erreichten MTIEs und Setup Zeiten feststellen. Schon bei der Analyse der Tracefiles (siehe 2.6) haben wir keine Unterschiede erkennen können.

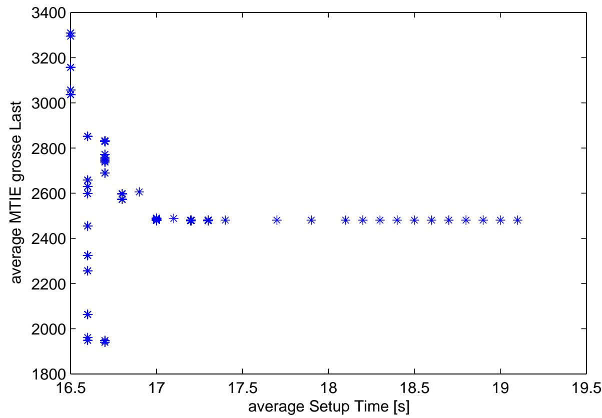

Abbildung 4.4: Zeigt die sehr hohen MTIE-Werte bei grosser Last in Relation zum MTIE bei kleiner Last beim LLR Algorithmus.

4.2.3 Szenarien

Zwischen dem Ad-Hoc und Infrastructure Szenario konnten wir keine signifikanten Unterschiede bei den erreichten MTIEs und Setup Zeiten feststellen.

Bei den beiden Ad-Hoc Szenarien mit Zeitnachrichten und Last in gleicher oder verschiedener Richtung (Abbildung 2.1) haben wir folgende Zahlen erhalten:

	Verschiedene Richtungen	Gleiche Richtung
MTIE klein	8,4 μs	9 μs
MTIE gross	10,2 μs	9,6 μs

Wie wir sehen, ist der MTIE klein besser bei verschiedener als bei gleicher Richtung. Das Umgekehrte ist der Fall bei MTIE gross. Wir sind uns bewusst, dass die Werte keinen grossen Unterschied aufweisen. Trotzdem können wir den kleinen Unterschied mit zwei Effekten erklären, die sich negativ auf den Delay der Zeitnachrichten auswirken:

- Bei verschiedenen Richtungen treten Kollisionen auf dem Shared Medium auf.

- Bei gleicher Richtung müssen die Zeitnachrichten im Sendepuffer warten.

Bei gleicher Richtung liessen sich die Werte noch verbessern, indem man mit einem Packet Scheduler die Zeitnachrichten gegenüber den Streaming Paketen priorisiert.

4.2.4 Guter MTIE auf Kosten hoher Setup Zeit

Als weiteres Resultat fanden wir heraus, dass ein guter MTIE bei kleiner Last einen guten MTIE bei grosser Last nicht ausschliesst. Die Simulation kann gut nach beiden Kriterien optimieren. Jedoch schliesst ein guter MTIE eine gute Setup Zeit aus, wie in Abbildung 4.5 zu sehen ist.

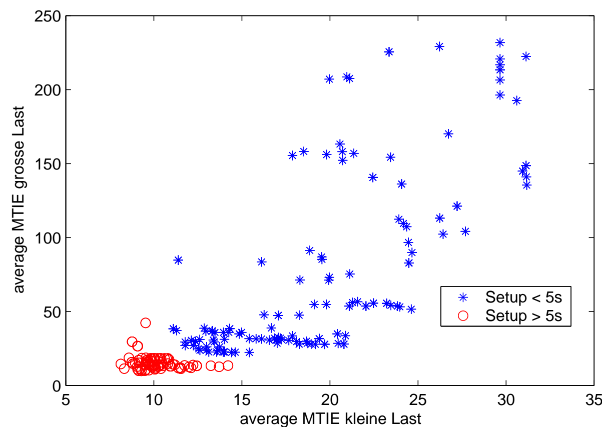

Abbildung 4.5: Alle Individuen mit gutem MTIE bei kleiner Last und MTIE bei grosser Last haben eine grosse Setup Zeit

4.2.5 Kriterien

Am Anfang legten wir uns auf den MTIE als Bewertungskriterium fest. Die Simulation ergab einen Fehler zur tatsächlichen Zeit wie in Abbildung 4.6 links. Er verläuft nicht in einem engen Band sondern schwankt langsam über die Zeit. Dieses Verhalten lässt einen guten MTIE zu, der jeweils nur den Abschnitt im Beobachtungsfenster berücksichtigt. Schliesslich haben wir ja auch nach MTIE optimiert.

Je nach Anwendung können diese langsamen Schwankungen aber auch störend sein. Darum haben wir noch eine Simulation mit einer Optimierung nach dem peak-to-peak Jitter durchgeführt. Der resultierende Errorvektor ist in Abbildung 4.6 rechts dargestellt. Hier verläuft der Fehler in einem engen Band von etwa $14\,\mu$s.

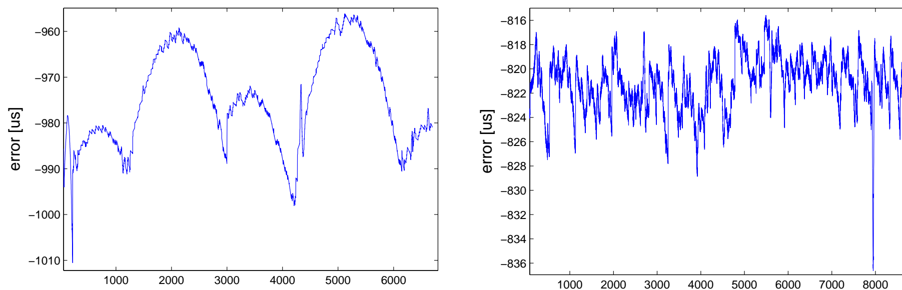

Abbildung 4.6: links: Error bei Optimierung nach MTIE; rechts: Error bei Optimierung nach Jitter. Die X-Achse entspricht etwa 30 Minuten.

Wie den untenstehenden Tabellen zu entnehmen ist, erreicht die Optimierung nach dem peak-to-peak Jitter schlechtere Werte als die Optimierung nach MTIE. Besonders die Möglichkeit, bei grosser Last längerfristige Driftschwankungen zuzulassen (Optimierung nach MTIE), führt zu kleineren kurzfristige Schwankungen.

Optimierung nach Jitter

	kleine Last [μs]	grosse Last [μs]	Setup Zeit [s]
Jitter	14	25	5,8
MTIE	10,8	14,7	2,2

Optimierung nach MTIE

	kleine Last [μs]	grosse Last [μs]	Setup Zeit [s]
MTIE	8,9	10,8	4
Jitter	22.6	171,2	7,7

Kapitel 5

Schlussfolgerungen

Wir konnten die Parameter von Synchronisationsalgorithmen mit evolutionären Methoden optimieren. Wir beschäftigten uns hauptsächlich mit dem LSDC Algorithmus und seinen sieben Parametern. Die gefundenen Parameter erzielen bessere Werte, als wenn man sie von Hand sucht. Vor allem gelang es uns Parameter zu finden, die sowohl bei kleiner als auch bei grosser und variierender Last ein sehr gutes Verhalten zeigen.

Unsere C-Implementierung für das PISA Framework funktioniert und liefert gute Optimierungen in vernünftiger Zeit. Vier Algorithmen sind implementiert und es können sehr leicht weitere hinzugefügt werden. Zudem liefert die Optimierung nach verschiedenen Kriterien die gewünschten Resultate. Auch die Output Funktionen der Simulation sind vielfältig. Man erhält eine gute Übersicht über den Ablauf und die Resultate der Simulation.

Hiermit bestätigen wir, dass wir die vorliegende Arbeit selber durchgeführt und verfasst haben.

Zürich, den 20. Februar 2004

Christian Hitz   Stefan Schuler

Anhang A

Bedienung der PISA Tools

Wir haben zwei Programme: `timesync` und `eval_param`. `timesync` macht die eigentliche Simulation und mit `eval_param` können gefundene Parametersätze auch auf anderen Szenarien ausgewertet werden. Zusätzlich kann `eval_param` noch Error, Drift, synchronisierte Zeit und Referenzzeit ausgeben.

Unsere Programme werden wie folgt gestartet:

```
timesync <paramfile> <filenamebase> <pollintervall>
eval_param <paramfile>
```

Das `<paramfile>` ist das Parameterfile im unten beschriebenen Format, `<filenamebase>` ist die Basis für die Files, um mit dem Selektor zu kommunizieren und `<pollintervall>` ist der Polling Abstand in Sekunden.

Einen passenden Selektor kann man von [1] herunterladen. Wir haben den Strength Pareto Evolutionary Algorithm 2 (SPEA2) Selektor verwendet.

Im folgenden wird die Struktur des Parameter-Files näher erläutert.

A.1 Simulationsablauf

Im Folgenden werden wir kurz einen Ablauf einer Simulation Schritt für Schritt durchgehen.

1. Parameterfile vorbereiten. Das beiliegende `timesync_param.txt` funktioniert hier.

2. Im File `lsdc6_cfg` die PISA Parameter anpassen. (`alpha`: Initialpopulation, `mu`: Anzahl Vorfahren pro Generation, `lambda`: Anzahl Nachkommen pro Generation, `dimension`: Anzahl Zielwerte)

3. Selektor von [1] herunterladen. Hier verwenden wir SPEA2.

4. SPEA2 entkomprimieren und `spea2` und `spea2_param.txt` ins Verzeichnis von `timesync` kopieren.

5. Zwei Shells starten und in beiden ins Verzeichnis von `timesync` wechseln.
6. Im ersten Shell mit folgendem Befehl `timesync` starten:

   ```
   ./timesync timesync_param.txt lsdc6_ 1
   ```

7. Im zweiten Shell mit folgendem Befehl `spea2` starten:

   ```
   ./spea2 spea2_param.txt lsdc6_ 1
   ```

Im ersten Shell können Sie nun den Verlauf der Simulation verfolgen.

A.2 Format des Parameterfiles

Das Parameter-File muss folgende Parameter in der vorgestellten Reihenfolge enthalten (siehe Beispiel in E.5).

MAX_GEN	Anzahl der Generationen.
NUMBER_TRACE_MESSAGES	Anzahl der Zeitstempel in den jeweiligen Tracefiles. Ist dieser Wert zu hoch, terminiert das Programm und meldet den maximal möglichen Wert.
NUMBER_TRACE_FILES	Anzahl der Tracefiles.
TRACE_FILE_NAME:	kein Wert.
<Tracefiles>	Die nächsten Zeilen enthalten jeweils den Pfad zu einem Tracefile.
OUTFILE_NAME	Filenamebase für die Output-Files. Directories werden generiert, falls nicht vorhanden.
ALGORITHM	0 = LSDC6, 1 = PLL2, 2 = LLR_CONT, 3 = GRADIENT.
NUMBER_OPT_PARAMETERS	Anzahl der Parameter, die optimiert werden sollen.
NUMBER_PARAMETERS	Gesamte Anzahl der Parameter.
EVAL_TIME	Auswertungszeitpunkt des Jitters in Sekunden.
MTIE_WINDOW	MTIE Fenstergrösse τ in Sekunden. ACHTUNG: wird auf das nächst kleinere 2^n Fenster abgerundet.
SAMPLES_INT	Abstand der Zeitstempel im Tracefile in Sekunden.

MUTATION_FACTOR	Stärke der Mutation in Prozent (0 - 1).
MTIE_SETUP	MTIE Setupzeit (MTIE wird erst danach berechnet) in Sekunden.
OPT_JITTER	1 = nach Jitter optimieren.
OPT_TVAR	1 = nach TVAR optimieren.
OPT_MTIE	1 = nach MTIE optimieren.
PRINT_ERROR	nur `eval_param`: 1 = Errorvektor, Drift, synchronisierte Zeit und Referenzzeit werden für alle Tracefiles ausgegeben.
<Parameter>	nur `eval_param`: Die zu evaluierenden Parameter. Im Exponentionalformat: `3.966024E-07`. Durch Leerzeichen, Tabstop oder Zeilenschaltung getrennt.

A.3 Einfügen eines neuen Algorithmus

Alle Funktionen zu Algorithmen und Auswertungskriterien sind im File `variator_user.c` implementiert. Um einen neuen Algorithmus einzufügen müssen folgende Schritte durchgeführt werden:

1. Implementierung der Initialisierungsfunktion in `variator_user.c` und deklarieren in `variator_user.h`.
2. Implementierung des neuen Algorithmus in `variator_user.c` und deklarieren in `variator_user.h`.
3. Ev. ein `enum` mit den Parameternamen definieren.
4. Einfügen der Initialisierungsfunktion im `switch` Statement der Funktion `state0()`.
5. Einfügen des Algorithmus im `switch` Statement in `calculate_ov()`.
6. Einfügen des Algorithmusnamen und der Parameternamen im `switch` Statement in `output()`.

Falls der neue Algorithmus mehr als 7 Parameter enthält, muss zusätzlich in `variator_user.h` die Konstante `MAX_NUMBER_PARAMETERS` angepasst werden.

A.4 Einfügen eines neuen Kriteriums

Das Einfügen eines neuen Kriteriums gestaltet sich etwas komplizierter als bei einem Algorithmus. Folgende Punkte müssen dabei beachtet werden:

- Implementierung des Kriteriums. Diese Funktion muss als Parameter mindestens den Errorvektor übernehmen und als Returnwert die Bewertung (als `double`) zurück geben. Allfällige weitere Rückgabewerte müssen via Referenz zurück gegeben werden (siehe `setup_time` bei `mtie()`).

- Einfügen der Kriteriumsfunktion in `calculate_ov()`. Falls das neue Kriterium kein altes komplett ersetzt, muss in `variator_user.h` im Struct `individual_t` ein weiteres Array definiert werden, sonst kann eine bereits bestehende Variable verwendet werden.

- Anpassen der Funktion `get_objective_value()`, damit das neue Kriterium auch bei der Simulation verwendet wird.

Anhang B

Projektbeschrieb

B.1 Einleitung

In Computer Netzwerken wird für verschiedene Zwecke eine allen Knoten gemeinsame und synchronisierte Zeit benötigt. Diese Aufgabe ist nicht trivial, da alle Knoten zwar eine Uhr besitzen (im PC z.B. die Real-time clock und der Prozessor Takt), diese aber mit leicht unterschiedlichen Geschwindigkeiten laufen. Daher müssen Zeitnachrichten zwischen Knoten ausgetauscht werden, welche von einem Synchronisationsalgorithmus ausgewertet werden. Als Resultat dieses Algorithmus wird dann die Zeit und/oder die Laufgeschwindigkeit der Uhr des entsprechenden Knotens leicht angepasst.

Die Genauigkeit, welche bei der Synchronisation von Uhren in einem Netzwerk erreicht werden kann, hängt natürlich vom verwendeten Algorithmus, aber auch stark von den Latenzen der verwendeten Zeitnachrichten ab. In vielen Netzwerken, z.B. auch in 802.11b Wireless LANs, sind diese Latenzen nicht normalverteilt und die Verteilung (PDF/CDF) ist auch stark lastabhängig. Daher liefert das künstliche Generieren von Nachrichten Laufzeiten meistens keine realistischen Werte. Um aussagekräftige Resultate zu erhalten, muss mit *gemessenen Latenz-Sequenzen* gearbeitet werden.

Um nun die Parameter eines bestimmten Synchronisationsalgorithmus zu optimieren, muss dessen Qualität für alle Varianten vergleichbar bestimmt werden. Absolute Vergleichbarkeit gibt es aber nur, wenn die Randbedingungen auch gleich bleiben. Dies wird durch Simulation erreicht, d.h. ein Algorithmus arbeitet nicht mit wirklichen Zeitnachrichten, sondern mit einem Trace File, in welchem die aufgezeichneten Zeitstempel abgespeichert sind. Obwohl es sich dabei um Simulation handelt, sind die Ergebnisse realistisch, da sie auf echt gemessenen Daten basieren. Mit diesem Vorgehen lassen sich also wiederholbare, realistische Bewertungen von Synchronistaions-algorithmen gewinnen.

Das oben beschriebene Verfahren wurde am Institut TIK entwickelt und ist in den Technical Reports [5] und [4] dokumentiert. Im Rahmen der hier

beschriebenen Arbeit soll das Verfahren ausgebaut und verbessert werden:

- Algorithmen sollen mit den evolutionären Methoden des ebenfalls am TIK entwickelten Frameworks *PISA* optimiert werden. Dazu müssen die bestehenden Algorithmen nach C portiert werden.
- Bisher wurde die Netzlast sehr einfach generiert, d.h. mit Prozessen, welche in konstanten Intervallen UDP-Pakete mit einer vorgegebenen Länge generieren. Diese Methode soll ersetzt werden durch echte Multimedia Anwendungen, welche MP3-Audio oder MPEG-Video Streams generieren.

Im Rahmen dieser Arbeit sollen dann verschiedene Messreihen aufgenommen und zur Optimierung verwendet werden. Die Resultate sollen nach wissenschaftlichen Kriterien ausgewertet werden. Folgende Fragen harren einer Antwort:

- Konvergiert die Optimierung in zumindest ähnlichen Parameterwerten für verschiedene Traces eines gleichen Szenarios? (Wenn nicht ist die ganze Optimierung sinnlos.)
- Falls ja, sind diese Parameter auch für verschieden Szenarios relativ vergleichbar? (Dann wäre der entsprechende Algorithmus robust.)
- Lässt sich die Synchronisierungs-Qualität quantitativ mit dem Szenario in Verbindung setzen? Wie sieht die Beziehung aus zwischen Last im Netz, bzw. der Frequenz der Zeitnachrichten und der erreichten Synchronisations-Qualität?

B.2 Aufgabenstellung

1. Machen Sie sich mit der Problemstellung vertaut. Nehmen Sie ihre Arbeitsrechner in Betrieb und installieren Sie die bestehenden Software Komponenten.
2. Machen Sie sich mit der Simulation von Synchronisierungsalgorithmen basierend auf den aufgenommenen Delay- und Driftmessungen vertraut. Gehen Sie dabei wie in [4], Anhang A beschrieben vor.
3. Machen Sie sich mit PISA vertraut. Studieren Sie dazu die Informationen und Beispiele von [3].
4. Erstellen Sie ein Konzept zur Optimierung mit PISA, sowie zur genetischen Kodierung der verschiedenen Synchronisierungsalgorithmen. Wie können letztere mutiert und rekombiniert werden? Überlegen Sie sich Kriterien nach denen Simulationsergebnisse bewertet werden. Setzen Sie das Konzept in C um.

5. Nehmen Sie manuell einige Delay- und Driftmessungen auf, um die bestehenden Software-Komponenten kennen zu lernen. Gehen Sie dabei wie in [5], Anhang A beschrieben vor. Überlegen Sie sich, ob dieser Vorgang automatisiert werden kann. Sprechen Sie mit dem Betreuer ab, wie weit dies sinnvoll ist.

6. Suchen Sie nach einer realistischen Generierung von Last im Netzwerk. Idealerweise verwenden Sie dazu eine bestehende Audio/Video Streaming Anwendung.

7. Nehmen Sie verschiedene Messreihen auf. Wenden Sie die PISA Optimierung auf diese Daten an. Versuchen Sie die Resultate sinnvoll darzustellen und auszuwerten. Welche Schlüsse ziehen Sie?

8. Dokumentieren Sie Ihre Arbeit sorgfältig mit einem Vortrag, einer Demonstration, sowie mit einem Schlussbericht.

B.3 Durchführung der Semesterarbeit

B.3.1 Allgemeines

- Erstellen Sie einen Projektplan und legen Sie Meilensteine fest [8]. Erarbeiten Sie in Absprache mit dem Betreuer ein Pflichtenheft.

- Stellen Sie Ihr Projekt zu Beginn der Semesterarbeit in einem Kurzvortrag vor und präsentieren Sie die erarbeiteten Resultate am Schluss im Rahmen des Institutskolloquiums Ende Semester.

- Besprechen Sie Ihr Vorgehen regelmässig (am besten fixer Termin einmal pro Woche) mit Ihrem Betreuer.

- Der Verlauf des Projektes soll laufend anhand des Projektplanes und der Meilensteine evaluiert werden. Unvorhergesehene Probleme beim eingeschlagenen Lösungsweg können Änderungen am Projektplan erforderlich machen. Diese sollen dokumentiert werden.

B.3.2 Material

- 4 PCs mit SuSE Linux, mit je einer IEEE 802.11b Wireless-LAN-Karte.

- Wireless Accesspoint.

- Parallel-Port-Verbindungskabel für die Delay-Messungen.

B.3.3 Abgabe

- Geben Sie vier unterschriebene Exemplare des Berichts spätestens am *20. Februar 2004* dem betreuenden Assistenten oder seinem Stellvertreter ab. Diese Aufgabenstellung soll im Bericht eingefügt werden. *Der Bericht soll auch sämtliche von ihnen geschriebene Software im Quelltext enthalten.* Besprechen Sie sich mit dem Betreuer, falls dies vom Umfang her nicht zweckmässig erscheint. Dem Bericht soll eine CD mit allen verwendeten Tools und den erstellten Programmen beiliegen.

- Räumen Sie Ihre Rechnerkonten soweit auf, dass nur noch die relevanten Source- und Objectfiles, Konfigurationsfiles, benötigte Directorystrukturen usw. bestehen bleiben. Der Programmcode sowie die Filestruktur soll ausreichend dokumentiert sein. Eine spätere Anschlussarbeit soll auf dem hinterlassenen Stand aufbauen können.

Anhang C

Zeitplan

Woche 43 20.10. – 24.10.	Problemstellung kennenlernen, Software installieren, PISA kennenlernen
Woche 44 27.10. – 31.10.	Konzept erarbeiten, Bericht schreiben mit LaTeX, MATLAB-Simulationen kennenlernen
Woche 45 3.11. – 7.11.	Portieren eines Algorithmus nach C
Woche 46 10.11. – 14.11.	Optimieren und Auswerten der Resultate mit PISA anhand Kriterien; 14.11. Präsentation
Woche 47 17.11. – 21.11.	Vorbereitung Delay-, Driftmessungen
Woche 48 24.11. – 28.11.	Christian: WK Stefan: Delay- und Driftmessungen
Woche 49 1.12. – 5.12.	Christian: WK Stefan: Delay- und Driftmessungen
Woche 50 8.12. – 12.12.	Messreihen aufnehmen
Woche 51 15.12. – 19.12.	Messreihen aufnehmen (Reserve)

Woche 2 5.1. – 9.1.	Vertiefung
Woche 3 12.1. – 16.1.	Vertiefung
Woche 4 19.1. – 23.1.	Vertiefung
Woche 5 26.1. – 30.1.	Resultate auswerten, Bericht schreiben mit LaTeX
Woche 6 2.2. – 6.2.	Resultate auswerten, Bericht schreiben mit LaTeX, Präsentation vorbereiten
Woche 7 9.2. – 13.2.	Arbeiten abschliessen, aufräumen Abgabe: 13.2.2004

Meilensteine:

- PISA Optimierungen
- Delay-, Driftmessungen

Vertiefung:

- Rekombination und Mutation der Individuen verbessern
- zusätzliche Zeitsynchronisations-Algorithmen portieren und optimieren
- Kriterien erweitern

Anhang D

Files auf CD

`Bericht/`	
`Bericht.tex`	Hauptsource im LaTeX Format
`*.tex`	Kapitelsourcen im LaTeX Format
`bibliographie.bib`	Referenzen im BibTeX Format
`Bericht.dvi`	Bericht im dvi Format
`Bericht.pdf`	Bericht im pdf Format
`Bericht.ps`	Bericht im PostScript Format
`figures/`	Graphiken für den Bericht
`Matlab/`	
`analyse_parameter.m`	Zeichnet die Zielwerte und die korrespondierenden Parameter untereinander.
`analyse_trace.m`	Plottet die Abstände zwischen benachbarten Zeitstempeln in Tracefiles.
`dominatePts.c`	C Implementation der Paretopunktsuche.
`dominatePts.mexglx`	`dominatePts.c` kompiliert für Linux i386.
`error_plot.m`	Plottet den Verlauf des Errorvektors.
`pareto_movie.m`	Generiert einen Film mit dem Verlauf der Paretofront.
`pareto_plot.m`	Zeichnet die Paretopunkte.
`paretoplot3d.m`	Zeichnet 3D-Plot der Paretopunkte.
`parsemain.m`	Normalisierung der Tracefiles.
`print_main.m`	Mittelt PDF, CDF und Delays über mehrere Tracefiles.

<table>
<tr><td> showmovie.m</td><td>Zeigt Paretofilm an.</td></tr>
<tr><td>PISA/</td><td></td></tr>
<tr><td> Results/</td><td>Einige Resultate von Simulationsläufen (gezippt) und 3 Paretofilme im QuickTime (.mov) Format.</td></tr>
<tr><td> Makefile</td><td>Makefile</td></tr>
<tr><td> timesync</td><td>Linux i386 binary</td></tr>
<tr><td> eval_param</td><td>Linux i386 binary</td></tr>
<tr><td> variator.c</td><td>\</td></tr>
<tr><td> variator.h</td><td>|</td></tr>
<tr><td> variator.o</td><td>|</td></tr>
<tr><td> variator_internal.c</td><td>|</td></tr>
<tr><td> variator_internal.h</td><td>|</td></tr>
<tr><td> variator_internal.o</td><td>| Sourcecode</td></tr>
<tr><td> variator_user.c</td><td>|</td></tr>
<tr><td> variator_user.h</td><td>|</td></tr>
<tr><td> variator_user.o</td><td>|</td></tr>
<tr><td> evaluate_parameter.c</td><td>|</td></tr>
<tr><td> evaluate_parameter.h</td><td>|</td></tr>
<tr><td> evaluate_parameter.o</td><td>/</td></tr>
<tr><td> variator_param.txt</td><td>Beispiel Parameterfile</td></tr>
<tr><td> lsdc6_cfg</td><td>PISA Kontrollfile</td></tr>
<tr><td>Tracefiles/</td><td></td></tr>
<tr><td> runTracesAdHoc.sh</td><td>Shell Skript um einen Durchlauf im Ad-Hoc Modus durchzuführen.</td></tr>
<tr><td> runTracesInfrastructure.sh</td><td>Shell Skript um einen Durchlauf im Infrastructure Modus durchzuführen.</td></tr>
<tr><td> AdHocGr20.tar.bz2</td><td>\</td></tr>
<tr><td> AdHocGr200.tar.bz2</td><td>|</td></tr>
<tr><td> AdHocGr2000.tar.bz2</td><td>|</td></tr>
<tr><td> AdHocVr20.tar.bz2</td><td>|</td></tr>
<tr><td> AdHocVr200.tar.bz2</td><td>|</td></tr>
<tr><td> AdHocVr2000.tar.bz2</td><td>| Jeweils 20 normalisierte</td></tr>
<tr><td> In20-3297.tar.bz2</td><td>| Tracefiles mit kleiner und</td></tr>
<tr><td> In20-3319.tar.bz2</td><td>| grosser Last.</td></tr>
<tr><td> In200-3297.tar.bz2</td><td>|</td></tr>
<tr><td> In200-3319.tar.bz2</td><td>|</td></tr>
<tr><td> In2000-3297.tar.bz2</td><td>|</td></tr>
<tr><td> In2000-3319.tar.bz2</td><td>/</td></tr>
<tr><td> tracefiles.tar.bz2</td><td>Alle Tracefiles (roh und normalisiert)</td></tr>
</table>

Anhang E

C Source Code

E.1 variator_user.c

```
/*============================================================
 * variator_user.c
 *
 * Implementation of the simulation for time synchrinisation
 * algorithms with evolutionary methods.
 *
 * 2003/2004: Christian Hitz, Stefan Schuler
 *
 * Framework provided by TIK
 *==========================================================*/
/*============================================================
  PISA  (www.tik.ee.ethz.ch/pisa/)
  ============================================================
  Computer Engineering (TIK)
  ETH Zurich
  ============================================================
  PISALIB
  Functions the user must implement
  C file.
  file: variator_user.c
  author: Fabian Landis, flandis@ee.ethz.ch
  last change: 20.01.03
  ============================================================
*/

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <libgen.h>
#include <errno.h>

#include "variator.h"
#include "variator_user.h"

char *log_file = "variator_error.log";
/**** Change the definition of 'log_file' if you want another
      name! Or set to NULL if you want to write error messages
      to stderr. */
/*-------------------| individual |------------------------*/

void free_individual(individual *ind)
/* Frees the memory for one indiviual.
   post: memory for ind is freed
*/
{
    /**** Here comes your code for freeing the memory
          allocated for the members of the 'individual_t'
          struct. */
    if(ind == NULL) return;

    free(ind);
}

double mtie_average(individual* temp_ind){
    int j;
    double mtie_average = 0;
    for(j = 0; j < NUMBER_TRACE_FILES/2; j++){
        mtie_average += temp_ind->mtie[j];
    }

    return mtie_average/(NUMBER_TRACE_FILES/2);
}

double mtie_average2(individual* temp_ind){
    int j;
    double mtie_average = 0;
    for(j = NUMBER_TRACE_FILES/2; j < NUMBER_TRACE_FILES; j++){
        mtie_average += temp_ind->mtie[j];
    }

    return mtie_average/(NUMBER_TRACE_FILES/2);
}

double setup_time_average(individual* temp_ind){
    int j;
    double setup_time_average = 0;
```

```
    for(j = 0; j < NUMBER_TRACE_FILES/2; j++){
        setup_time_average += temp_ind->setup_time[j];
    }
    return setup_time_average/(NUMBER_TRACE_FILES/2);
}

double jitter_average(individual* temp_ind){
    int j;
    double jitter_average = 0;
    for(j = 0; j < NUMBER_TRACE_FILES/2; j++){
        jitter_average += temp_ind->jitter[j];
    }

    return jitter_average/(NUMBER_TRACE_FILES/2);
}

double jitter_average2(individual* temp_ind){
    int j;
    double jitter_average = 0;
    for(j = NUMBER_TRACE_FILES/2; j < NUMBER_TRACE_FILES; j++){
        jitter_average += temp_ind->jitter[j];
    }

    return jitter_average/(NUMBER_TRACE_FILES/2);
}

double setup_time_j_average(individual* temp_ind){
    int j;
    double setup_time_j_average = 0;
    for(j = 0; j < NUMBER_TRACE_FILES/2; j++){
        setup_time_j_average += temp_ind->setup_time_j[j];
    }
    return setup_time_j_average/(NUMBER_TRACE_FILES/2);
}

double worse(individual* temp_ind){
    int j;
    double worse = 0;
    for(j = 0; j < NUMBER_TRACE_FILES/2; j++){
        worse += (temp_ind->mtie[j+NUMBER_TRACE_FILES/2])/
                                        (temp_ind->mtie[j]);
    }
    return worse/(NUMBER_TRACE_FILES/2);
}

double get_objective_value(int identity, int i)
/* Gets objective value of an individual.

pre: 0 <= i <= dimension - 1 (dimension is the number of
```

```
                                objectives)

post: Return value == the objective value number 'i' in
      individual '*ind'.
      If no individual with ID 'identity' return value == -1.
*/
{
    double objective_value = -1.0;
    individual *temp_ind;
    /* asserting the pre-condition */
    assert(0 <= i && i < dimension);

    /**** Here comes your code for getting the objective
          values from the 'individual_t' struct. */

    temp_ind = get_individual(identity);
    if (temp_ind == NULL) return (-1);

    if(OPT_MTIE && !OPT_JITTER && !OPT_TVAR){
        if(i == 0) objective_value = mtie_average(temp_ind);
        else if(i == 1) objective_value = mtie_average2(temp_ind);
        else if(i == 2) objective_value = setup_time_average(temp_ind);
        else return (-1);
    }
    else if(OPT_JITTER && !OPT_MTIE && !OPT_TVAR){
        if(i == 0) objective_value = jitter_average(temp_ind);
        else if(i == 1) objective_value = jitter_average2(temp_ind);
        else if(i == 2) objective_value = setup_time_j_average(temp_ind);
        else return (-1);
    }
    else if(!OPT_MTIE && OPT_TVAR){
        objective_value = temp_ind->tvar[i];
    }
    else{
        if(i < NUMBER_TRACE_FILES) objective_value = temp_ind->mtie[i];
        else if(i < 2*NUMBER_TRACE_FILES) objective_value =
                             temp_ind->setup_time[i-NUMBER_TRACE_FILES];
        else objective_value = temp_ind->tvar[i-2*NUMBER_TRACE_FILES];
    }

    return (objective_value);
}

/*-------------------| statemachine functions |-------------*/
int state0()
/* Do what needs to be done in state 0.

pre: The global variable 'paramfile' contains the name of the
     parameter file specified on the commandline.
```

```
     The global variable 'alpha' contains the number of indiviuals
     you need to generate for the initial population.

post: Optionally read parameter specific for the module.
      Optionally do some initialization.
      Initial population created.
      Information about initial population written to the ini file
      using write_ini().
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{

    int result; /* stores return values of called functions */
    int *initial_population; /* storing the IDs of the individuals */
    int i, j;
    char inter_out[FILE_NAME_LENGTH], inter_out2[10][FILE_NAME_LENGTH], *dir[10];

    initial_population = malloc(alpha * sizeof(int));
    if (initial_population == NULL)
    {
        log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
        return (1);
    }

    /**** Here you could call a function to read the local parameter
          values from the 'paramfile'. */
    /* strcpy(paramfile,"variator_param.txt"); */
    if(read_local_parameters()) printf("error reading tracefile file\n");

    gen = 0;

    local_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                             NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(local_time != NULL);
    remote_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                             NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(remote_time != NULL);
    reference_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                             NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(reference_time != NULL);

    for(i = 0; i < NUMBER_TRACE_FILES; i++){
        read_tracefile(TRACE_FILE_NAME[i], &local_time[i*NUMBER_TRACE_MESSAGES],
                       &remote_time[i*NUMBER_TRACE_MESSAGES],
                       &reference_time[i*NUMBER_TRACE_MESSAGES]);
    }
```

```
/**** Here comes your code for creating the initial population.
      Add each individual to the global population using add_individual
      and put all identities in the 'initial_population' array. */
switch (ALGORITHM) {
    case 0:
        init_population_lsdc(initial_population);
        break;
    case 1:
        init_population_pll(initial_population);
        break;
    case 2:
        init_population_llr(initial_population);
        break;
    case 3:
        init_population_mpeg(initial_population);
        break;
}

/* Make recursivly all output directories if needed */
strcpy(inter_out2[0], OUTFILE_NAME);
j = 1;
do{
    dir[j] = dirname(inter_out2[0]);
    strcpy(inter_out2[j], dir[j]);
    printf("%d: %s\n", j, dir[j]);
}while((dir[j][0] != '.') && (j++ < 10));

for(i = j-1; i > 0; i--){
    if(mkdir(inter_out2[i], 0766) != 0){
        if(errno != EEXIST){ /* no error when dir already exists */
            perror("Creating outdir failed");
            exit(1);
        }
    }
}

sprintf(inter_out, "%s.in.txt", OUTFILE_NAME);
output(inter_out);

sprintf(inter_out, "%s.%d.txt", OUTFILE_NAME, gen);
output2(inter_out);

result = write_ini(initial_population);
if (result != 0)
{
    log_to_file(log_file, __FILE__, __LINE__,
                "couldn't write ini");
    free(initial_population);
    return (1);
```

```
    }
    gen++;
    free(initial_population);

    return (0);
}

int state2()
/* Do what needs to be done in state 2.

pre: The global variable 'mu' contains the number of indiviuals
     you need to read using 'read_sel()'.
     The global variable 'lambda' contains the number of individuals
     you need to create by variation of the individuals specified the
     'sel' file.

post: Optionally call read_arc() in order to delete old uncessary
      individuals from the global population.
      read_sel() called
      'lambda' children generated from the 'mu' parents
      Children added to the global population using add_individual().
      Information about children written to the 'var' file using
      write_var().
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{
    int *parent_identities, *offspring_identities; /* array of identities */
    int result; /* stores return values of called functions */
    char inter_out[FILE_NAME_LENGTH];

    parent_identities = malloc(mu * sizeof(int));
    if (parent_identities == NULL)
    {
        log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
        return (1);
    }

    offspring_identities = malloc(lambda * sizeof(int));
    if (offspring_identities == NULL)
    {
        log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
        return (1);
    }

    result = read_sel(parent_identities);
    if (result != 0) /* if some file reading error occurs, return 2 */
        return (2);
```

```
    result = read_arc();
    if (result != 0) /* if some file reading error occurs, return 2 */
        return (2);

    /**** Here comes your code for creating lambda new individuals by
          variating the parents in 'parent_identities'.
          add all inidividuals to the global population using
          add_individual() and write their IDs in offspring_identities. */
    evaluate(parent_identities,offspring_identities);

    sprintf(inter_out, "%s.%d.txt", OUTFILE_NAME, gen);
    output2(inter_out);

    result = write_var(offspring_identities);
    if (result != 0)
    {
        log_to_file(log_file, __FILE__, __LINE__,
                    "couldn't write var");
        free(offspring_identities);
        free(parent_identities);
        return (1);
    }
    gen++;
    free(offspring_identities);
    free(parent_identities);
    return (0);
}

int state4()
/* Do what needs to be done in state 4.

pre: State 4 means the variator has to terminate.

post: Free all memory.
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{
    /**** Here comes your code if you need to do anything before
          terminating. E.g. call some kind of output function, which
          informs about the results of the optimization run. */
    char inter_out[FILE_NAME_LENGTH];
    sprintf(inter_out, "%s.out.txt", OUTFILE_NAME);
    output(inter_out);
    free(local_time);
    free(remote_time);
    free(reference_time);
```

```
    return (0);
}

int state7()
/* Do what needs to be done in state 7.

pre: State 7 means that the selector has just terminated.

post: You probably don't need to do anything, just return 0.
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{
    return(0);
}

int state8()
/* Do what needs to be done in state 8.

pre: State 8 means that the variator needs to reset and get ready to
     start again in state 0.

post: Get ready to start again in state 0.
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{
    free(local_time);
    free(remote_time);
    free(reference_time);
    return (0);
}

int state11()
/* Do what needs to be done in state 11.

pre: State 11 means that the selector has just reset and is ready
     to start again in state 1.

post: You probably don't need to do anything, just return 0.
      Return value == 0 if successful,
                   == 1 if unspecified errors happened,
                   == 2 if file reading failed.
*/
{
    return (0);
}
```

```
int is_finished()
/* Tests if ending criterion of your algorithm applies.

post: return value == 1 if optimization should stop
      return value == 0 if optimization should continue
*/
{
    /**** Here comes your code for checking the termination criteria. */
    printf("Generation: %d\n",gen);
    return (gen > MAX_GEN);
}

#define max(x,y) (x>y)?x:y
#define min(x,y) (x<y)?x:y

/***** START Algorithmus-Spezifisch *****/
/* LSDC 6                              */
enum{  DRIFT_VAR_INIT,
       DRIFT_VAR_FACTOR,
       DRIFT_VAR_FINAL,
       INIT_PHASE,
       WINDOW_SIZE,
       ESTIM_ERROR,
       DRIFT_MAX};

/* creates alpha new individuals for the initial population */
void init_population_lsdc(int * initial_population){
    int i;
    individual * new_ind;

    /* set seed for random function to current system
       time in seconds */
    srand(time(0));

    for (i = 0; i < alpha; i++){
        new_ind = malloc(sizeof(individual));

        /* Initialwerte wurden optimiert nach langer PISA Simulation
           im AdHoc und Infrastructure Mode */
        /* uniformly distributed between 1 and 10 */
        new_ind->parameters[INIT_PHASE] =
                    1+(10-1)*(double)rand()/RAND_MAX;
        new_ind->parameters[WINDOW_SIZE] =
                    10+(1000-10)*(double)rand()/RAND_MAX;
        new_ind->parameters[ESTIM_ERROR] =
                    (10+(1000-10)*(double)rand()/RAND_MAX)*1e-6;
        new_ind->parameters[DRIFT_VAR_FACTOR] =
                    1+(10-1)*(double)rand()/RAND_MAX;
```

```
        new_ind->parameters[DRIFT_MAX] = 1e-3;
        new_ind->parameters[DRIFT_VAR_INIT] =
                    (1+(50-1)*(double)rand()/RAND_MAX)*1e-7;
        new_ind->parameters[DRIFT_VAR_FINAL] =
                    (1+(100-1)*(double)rand()/RAND_MAX)*1e-13;

        /* add new individual to inital population of the selector */
        initial_population[i] = add_individual(new_ind);
        calculate_ov(initial_population[i]);
    }
}
int lsdc6(int step, double *parameters, unsigned int *local_time,
          unsigned int *remote_time, double *ts, double *drift){
    static double estim_error;
    static double drift_var;
    unsigned int leap;

    if (step == 0){
        ts[0] = remote_time[0];

        estim_error = parameters[ESTIM_ERROR];
        drift_var = parameters[DRIFT_VAR_INIT];
    }
    else{
        ts[step] = ts[step-1]+(local_time[step]-local_time[step-1])/
                                        (1+drift[step-1]);

        if(remote_time[step] > ts[step]){
            leap = remote_time[step]-ts[step];
            ts[step] = remote_time[step];
        }
        else{
            leap = 0;
        }

        if(step > parameters[INIT_PHASE]){
            if(leap > 0){
                drift[step] = (local_time[step] -
                 local_time[max(1,step - (int)parameters[WINDOW_SIZE])])/
                 (ts[step] - ts[max(1,step-(int)parameters[WINDOW_SIZE])]
                                                  - estim_error) - 1;
                if(estim_error > 0){
                    estim_error = estim_error - 1e-6;
                }
                if(drift_var > parameters[DRIFT_VAR_FINAL]){
                    drift_var = drift_var / parameters[DRIFT_VAR_FACTOR];
                }
            }
            else{
```

```
                drift[step] = drift[step-1] + drift_var*(ts[step] - ts[step-1]);
            }
        }
        if(drift[step] < -1*parameters[DRIFT_MAX]){
            drift[step] =  -1*parameters[DRIFT_MAX];
        }
        if(drift[step] > parameters[DRIFT_MAX]){
            drift[step] =  parameters[DRIFT_MAX];
        }
    }
    return (0);
}
/***** ENDE Algorithmus-Spezifisch *****/

/***** START Algorithmus-Spezifisch *****/
/* LLR_CONT                             */
enum{ BUFFER_SIZE};

/* creates alpha new individuals for the initial population */
void init_population_llr(int * initial_population){
    int i;
    individual * new_ind;
    /* set seed for random function to current system
       time in seconds */
    srand(time(0));

    for (i = 0; i < alpha; i++){
        new_ind = malloc(sizeof(individual));
        new_ind->parameters[BUFFER_SIZE] =
                    1+(5000-1)*(double)rand()/RAND_MAX;

        /* add new individual to inital population of the selector */
        initial_population[i] = add_individual(new_ind);
        calculate_ov(initial_population[i]);
    }
}
int llr_cont(double *parameters, unsigned int *local_time,
             unsigned int *remote_time, double *t, double *rho){
    double sum_loc = 0,
           sum_ref = 0,
           sum_loc2 = 0,
           sum_loc_ref = 0,
           delta,
           offset,
           slope;
    int step, buffer;

    buffer = (int)parameters[BUFFER_SIZE];
```

```
    for(step = 0; step < NUMBER_TRACE_MESSAGES; step++){
        sum_loc += local_time[step];
        sum_ref += remote_time[step];
        sum_loc2 += local_time[step] * local_time[step];
        sum_loc_ref += local_time[step] * remote_time[step];
        if(step > buffer){
            sum_loc -= local_time[step-buffer];
            sum_ref -= remote_time[step-buffer];
            sum_loc2 -= local_time[step-buffer] * local_time[step-buffer];
            sum_loc_ref -= local_time[step-buffer] * remote_time[step-buffer];
        }

        delta = sum_loc2 - sum_loc * sum_loc;
        if(delta != 0){
            offset = (sum_loc2 * sum_ref - sum_loc * sum_loc_ref) / delta;
            slope = (sum_loc_ref - sum_loc * sum_ref) / delta;
            t[step] = offset + slope * local_time[step];
            rho[step] = 1 - slope;
        }
        else{
            t[step] = remote_time[step];
            rho[step] = 0;
        }
    }
    return (0);
}
/***** ENDE Algorithmus-Spezifisch *****/

/***** START Algorithmus-Spezifisch *****/
/* GRD_MPEG                             */
enum{  IOTA,
       GAMMA};

/* creates alpha new individuals for the initial population */
void init_population_mpeg(int * initial_population){
    int i;
    individual * new_ind;
    /* set seed for random function to current system
       time in seconds */
    srand(time(0));

    for (i = 0; i < alpha; i++){
        new_ind = malloc(sizeof(individual));
        new_ind->parameters[IOTA] = (1+(1000-1)*(double)rand()/RAND_MAX)*1e3;
        new_ind->parameters[GAMMA] = 0+(10-0)*(double)rand()/RAND_MAX;

        /* add new individual to inital population of the selector */
        initial_population[i] = add_individual(new_ind);
        calculate_ov(initial_population[i]);
```

```
    }
}
int mpeg_grad(double *parameters, unsigned int *local_time,
              unsigned int *remote_time, double *t, double *rho){
    double sum_loc = 0,
           sum_ref = 0,
           diff_loc = 0,
           diff_ref = 0,
           avg_loc = 0,
           avg_ref = 0;
    int step;
    int iota = (int)parameters[IOTA];

    for(step = 1; step <= NUMBER_TRACE_MESSAGES; step++){

        /* update predictor */
        if(step > 1){
            avg_loc = sum_loc / (step - 1);
            avg_ref = sum_ref / (step - 1);
            if ((step - 1) > parameters[GAMMA]){
                diff_loc = (iota - 1) / (float)iota * diff_loc +
                           1 / (float)iota * (local_time[step-1] - avg_loc);
                diff_ref = (iota - 1) / (float)iota * diff_ref +
                           1 / (float)iota * (remote_time[step-1] - avg_ref);
            }
        }
        sum_loc += local_time[step-1];
        sum_ref += remote_time[step-1];

        /*predict*/
        t[step-1] = sum_ref / step;
        avg_loc = sum_loc / step;
        if (diff_loc > 0){
            rho[step-1] = diff_loc / diff_ref - 1;
        }
        else{
            rho[step-1] = 0;
        }
        t[step-1] += (local_time[step-1] - avg_loc) / (1 + rho[step-1]);
    }
    return (0);
}
/***** ENDE Algorithmus-Spezifisch *****/

/***** START Algorithmus-Spezifisch *****/
/* PLL                                  */
enum{  LIMIT,
       K_I,
       K_P};
```

```
/* creates alpha new individuals for the initial population */
void init_population_pll(int * initial_population){
    int i;
    individual * new_ind;
    /* set seed for random function to current system
       time in seconds */
    srand(time(0));

    for (i = 0; i < alpha; i++){
        new_ind = malloc(sizeof(individual));
        /* parameter ranges based on a PISA-Run with high (0.3)
           mutation rate */
        new_ind->parameters[LIMIT] =
                    0+(100-0)*(double)rand()/RAND_MAX;
        new_ind->parameters[K_I] =
                    (1+(100-1)*(double)rand()/RAND_MAX) * 1e-15;
        new_ind->parameters[K_P] =
                    (1+(100-1)*(double)rand()/RAND_MAX) * 1e-8;

        /* add new individual to inital population of the selector */
        initial_population[i] = add_individual(new_ind);
        calculate_ov(initial_population[i]);
    }
}
int pll2(double *parameters, unsigned int *local_time,
         unsigned int *remote_time, double *t, double *rho){
    int step;
    double *offset = (double *)malloc(NUMBER_TRACE_MESSAGES * sizeof(double));
    double *I = (double *)malloc(NUMBER_TRACE_MESSAGES * sizeof(double));
    double theta, limit, k_i, k_p;

    limit = parameters[LIMIT];
    k_i = parameters[K_I];
    k_p = parameters[K_P];

    I[0] = 0;
    rho[0] = 0;
    offset[0] = remote_time[0] - local_time[0];
    t[0] = local_time[0] + offset[0];

    /* step thourgh the received timestamps */
    for(step = 1; step < NUMBER_TRACE_MESSAGES; step++){
        /* compute synchronized time based on previously computed drift
           and offset */
        t[step] = (1 + rho[step - 1]) * local_time[step] + offset[step - 1];
        /* compute input to pll: difference from new ref and actual
           sync-time */
        theta = remote_time[step] - t[step];
```

```
        if(theta > limit){
            theta = limit;
        }
        if(theta < -limit){
            theta = -limit;
        }
        /* PI-controller for clock drift */
        I[step] = I[step - 1] + k_i * theta *
                  (local_time[step] - local_time[step - 1]);
        rho[step] = k_p * theta + I[step];
        /* adjust offset such that sync-time is continuous */
        offset[step] = t[step] - (1 + rho[step]) * local_time[step];
    }
    free(offset);
    free(I);
    return (0);
}
/***** ENDE Algorithmus-Spezifisch *****/

void evaluate(int *parent_identities, int *offspring_identities){
    /* create variables and allocate memory then read tracefile */
    int index, k,i;
    double sel_mut_rec;

    /* recombination and mutation */
    for (k = 0; k < lambda; k++){
        individual *offspring = (individual*)malloc(sizeof(individual));
        individual *parent1 = get_individual(parent_identities[
                              (int)floor((double)rand()/RAND_MAX*(mu-1e-9))]);
        individual *parent2 = get_individual(parent_identities[
                              (int)floor((double)rand()/RAND_MAX*(mu-1e-9))]);
        memcpy(offspring, parent1, sizeof(individual));

        /* Mutation eines Parameters */
        sel_mut_rec = (double)rand()/RAND_MAX;
        if (sel_mut_rec <= 0.5){
            index = floor((double)rand()/
                          RAND_MAX*(NUMBER_OPT_PARAMETERS-1e-9));
            offspring->parameters[index] = parent1->parameters[index]*
                                 ((1-MUTATION_FACTOR) + (double)rand()/
                                           RAND_MAX*(2*MUTATION_FACTOR));
        }
        /* Rekombination */
        else{
            for(index = 0; index < NUMBER_OPT_PARAMETERS; index++){
                if (0.5 <= (double)rand()/RAND_MAX){
                    /* erbt Parameter von parent2 */
                    offspring->parameters[index] = parent2->parameters[index];
                }
```

```
            }
        }
        /* add mutated/recombined individuals to selector */
        offspring_identities[k] = add_individual(offspring);
    }

    for(i = 0; i < lambda; i++){
        calculate_ov(offspring_identities[i]);
    }
}

void calculate_ov(int ind){
    double *synchronized_time, *drift, *error;
    individual *actualInd;
    int i,j, result, setup_time;
    char out[FILE_NAME_LENGTH], out2[FILE_NAME_LENGTH];
    /* calculate objective values */
    synchronized_time = (double*)malloc(NUMBER_TRACE_MESSAGES*sizeof(double));
    drift = (double*)malloc(NUMBER_TRACE_MESSAGES*sizeof(double));
    error = (double*)malloc(NUMBER_TRACE_MESSAGES*sizeof(double));

    actualInd = get_individual(ind);

    for(i = 0; i < NUMBER_TRACE_FILES; i++){
switch (ALGORITHM) {
    case 0:
for(j = 0; j < NUMBER_TRACE_MESSAGES; j++){
    result = lsdc6(j, actualInd->parameters,
   &local_time[i*NUMBER_TRACE_MESSAGES],
   &remote_time[i*NUMBER_TRACE_MESSAGES],
   synchronized_time, drift);
}
break;
    case 1:
result = pll2(actualInd->parameters,
      &local_time[i*NUMBER_TRACE_MESSAGES],
      &remote_time[i*NUMBER_TRACE_MESSAGES],
      synchronized_time, drift);
break;
    case 2:
result = llr_cont(actualInd->parameters,
  &local_time[i*NUMBER_TRACE_MESSAGES],
  &remote_time[i*NUMBER_TRACE_MESSAGES],
  synchronized_time, drift);
break;
    case 3:
result = mpeg_grad(actualInd->parameters,
   &local_time[i*NUMBER_TRACE_MESSAGES],
   &remote_time[i*NUMBER_TRACE_MESSAGES],
```

```
   synchronized_time, drift);
break;
}
        compute_error(error, synchronized_time,
                      &reference_time[i*NUMBER_TRACE_MESSAGES]);
        if(PRINT_ERROR == 1){
            sprintf(out, "%s.%d", OUTFILE_NAME, i+1);
            sprintf(out2, "%s.error.txt", out);
            output_error(out2, error);
            sprintf(out2, "%s.ts.txt", out);
            output_error(out2, synchronized_time);
            sprintf(out2, "%s.lt.txt", out);
            output_error2(out2, &reference_time[i*NUMBER_TRACE_MESSAGES]);
            sprintf(out2, "%s.drift.txt", out);
            output_error(out2, drift);
        }
        if(OPT_JITTER){
            actualInd->jitter[i] = jitter(&reference_time[i*NUMBER_TRACE_MESSAGES],
                                error, &setup_time, EVAL_TIME);
            actualInd->setup_time_j[i] = setup_time * SAMPLES_INT;
        }
/*      jitterslow(&reference_time[i*NUMBER_TRACE_MESSAGES], error, EVAL_TIME); */
/*      if(OPT_MTIE) actualInd->mtie[i] = mtieslow(error, MTIE_WINDOW/SAMPLES_INT,
                                NUMBER_TRACE_MESSAGES, MTIE_SETUP/SAMPLES_INT); */
        if(OPT_MTIE){
            actualInd->mtie[i] = mtie((int)(log(MTIE_WINDOW/SAMPLES_INT)/log(2)),
                               error, &setup_time);
            actualInd->setup_time[i] = setup_time * SAMPLES_INT;
        }
/*      actualInd->jitter = tvarslow(error, MTIE_WINDOW/SAMPLES_INT,
                                NUMBER_TRACE_MESSAGES, MTIE_SETUP/SAMPLES_INT); */
        if(OPT_TVAR) actualInd->tvar[i] = tvar(error, MTIE_WINDOW/SAMPLES_INT,
                               NUMBER_TRACE_MESSAGES, MTIE_SETUP/SAMPLES_INT);
    }

    free(error);
    free(synchronized_time);
    free(drift);
}

void compute_error(double *error, double* synchronized_time,
                   unsigned int* reference_time){
    int i;
    for(i = 0; i < NUMBER_TRACE_MESSAGES; i++){
        error[i] = synchronized_time[i]-reference_time[i];

    }
}
```

```
/* Jitter */
double jitterslow(unsigned int *reference_time,
                  double* error, int eval_time){
    double jitter, weight, weight_sum, jitter_sum;
    int i;
    weight_sum = 0;
    jitter_sum = 0;

    for(i = 0; i < NUMBER_TRACE_MESSAGES; i++){
        jitter = max_array(error, i, NUMBER_TRACE_MESSAGES-1) -
                 min_array(error, i, NUMBER_TRACE_MESSAGES-1);
        if(reference_time[i] < eval_time*1e6){
            weight = reference_time[i]/(eval_time*1e6);
        }
        else{
            weight = 1;
        }
        weight_sum += weight;
        jitter_sum += weight*jitter;
    }
/*     printf("jitterslow: %f\n", jitter_sum/weight_sum); */
    return jitter_sum/weight_sum;
}

double jitter(unsigned int *reference_time, double* error,
              int* setup_time, int eval_time){
    double jitter, weight, weight_sum, jitter_sum,
           setup_threshold, *setup_search;
    int i, setup_mark = 0, first = 1;
    struct{
        double *error;
        double previous_result;
    } minS, maxS;

    setup_search = (double*)calloc(NUMBER_TRACE_MESSAGES,
                                   sizeof(double));

    minS.error = error;
    minS.previous_result = minS.error[NUMBER_TRACE_MESSAGES-1];
    maxS.error = error;
    maxS.previous_result = maxS.error[NUMBER_TRACE_MESSAGES-1];

    weight_sum = 0;
    jitter_sum = 0;

    for(i = (NUMBER_TRACE_MESSAGES-1); i > -1; i--){
        maxS.previous_result = max(maxS.error[i],maxS.previous_result);
        minS.previous_result = min(minS.error[i],minS.previous_result);
        jitter = maxS.previous_result - minS.previous_result;
```

```
        if(reference_time[i] < eval_time*1e6){
            weight = 0; /* reference_time[i]/(eval_time*1e6); */
            setup_search[i] = jitter;
            if(first == 1){
                setup_mark = i;
                first = 0;
            }
        }
        else{
            weight = 1;
        }

        weight_sum += weight;
        jitter_sum += weight*jitter;
    }

    setup_threshold = 0.1 * (setup_search[0] - setup_search[setup_mark]);
    for(i = setup_mark; i >= 0; i--){
        if(setup_search[i] > setup_threshold){
            *setup_time = i;
            break;
        }
    }
    free(setup_search);

    /* printf("jitter calculated: %f\n",jitter_sum/weight_sum); */
    jitter = jitter_sum/weight_sum;

/*     printf("jitter:   %f6\n", jitter); */
    return jitter_sum/weight_sum;
}

double max_array(double *array, int i, int j){
    double high = array[i++];
    for(; i <= j; i++){
        high = max(high, array[i]);
    }
    return high;
}

double min_array(double *array, int i, int j){
    double high = array[i++];
    for(; i <= j; i++){
        high = min(high, array[i]);
    }
    return high;
}
/* END Jitter */
```

```
/* TVAR (Time Variance)*/
/* Measurement for time stability based on the modified Allan Variance */
double tvarslow(double *error, int window, int samples, int setup){
    int i,j;
    double Tj = 0;
    double Tj_sum = 0;
    double *error2;
    error2 = (double*)malloc((samples-setup)*sizeof(double));
    for(i = setup; i<samples; i++){
        error2[i-setup] = error[i];
    }
    samples = samples-setup;
    for(j = 0; j < samples-3*window+1; j++){
        Tj = 0;
        for(i = j; i <= window+j-1; i++){
            Tj += error2[i+2*window] - 2*error2[i+window] + error2[i];
        }
        Tj_sum += Tj*Tj;
    }
    Tj_sum = Tj_sum/(6*window*window);
    Tj_sum = Tj_sum/(samples - 3*window+1);
    return (Tj_sum);
}

double tvar(double *error, int window, int samples, int setup){
    int i,j;
    double Tj = 0;
    double Tj_sum = 0;
    for (i=setup; i <= window+setup-1; i++){
        Tj += error[i+2*window] - 2*error[i+window] + error[i];
    }

    for (j=setup; j < samples-3*window; j++){
        Tj_sum += Tj*Tj;
        /*calculate Tj recursively*/
        Tj += error[3*window+j] - 3*error[2*window+j] + 3*error[window+j] - error[j];
    }
    Tj_sum += Tj*Tj;
    Tj_sum = Tj_sum/(6*window*window);
    Tj_sum = Tj_sum/(samples - setup - 3*window+1);
    return (Tj_sum);
}
/* End TVAR */

/* MTIE (Maximum Time Interval Error) */
/* Maximum peak-to-peak difference in all possible window frames */
/* Complexity of order N^2*/
double mtieslow(double *error, int window, int samples, int setup){
    int i;
```

```
    double highest = 0;

    i = samples;
    for(i = setup; i < samples-window; i++){
        highest = max(highest, max_array(error,i,i+window-1) -
                               min_array(error,i,i+window-1));
    }
    return highest;
}

/* Complexity of order NlogN using binary Decomposition */
double mtie(int log2window, double* error, int* setup_time){
    double steady_state = 0, setup = 0, setup2 = 0, mtie_threshold;
    int i, j, size, jtmp, indextmp;
    double *max_mat, *min_mat;

    size = NUMBER_TRACE_MESSAGES;
    max_mat = (double*)calloc(sizeof(double), log2window*size);
    min_mat = (double*)calloc(sizeof(double), log2window*size);
    for(i = 0; i < NUMBER_TRACE_MESSAGES-1; i++){
        max_mat[i] = max(error[i], error[i+1]);
        min_mat[i] = min(error[i], error[i+1]);
    }
    for(j = 1; j < log2window; j++){
        jtmp = (int)pow(2,j);
        indextmp = (j-1)*size;
        for(i = 0; i < NUMBER_TRACE_MESSAGES-jtmp*2+1; i++){
            max_mat[i+indextmp+size] = max(max_mat[i+indextmp],
                                           max_mat[i+jtmp+indextmp]);
            min_mat[i+indextmp+size] = min(min_mat[i+indextmp],
                                           min_mat[i+jtmp+indextmp]);
        }
    }

    jtmp = NUMBER_TRACE_MESSAGES-(int)pow(2,log2window)+1;
    for(i = NUMBER_TRACE_MESSAGES/3; i < jtmp; i++){
        steady_state = max(steady_state, max_mat[i+(log2window-1)*size] -
                                         min_mat[i+(log2window-1)*size]);
    }

    setup = steady_state;
    for(i = NUMBER_TRACE_MESSAGES/3-1; i >= 0; i--){
        setup = max(setup, max_mat[i+(log2window-1)*size] -
                           min_mat[i+(log2window-1)*size]);
    }
    mtie_threshold = (setup - steady_state)*0.1;
    setup2 = steady_state;
    for(i = NUMBER_TRACE_MESSAGES/3-1; i >= 0; i--){
        setup2 = max(setup2, max_mat[i+(log2window-1)*size] -
```

```
                            min_mat[i+(log2window-1)*size]);
        if(setup2 - steady_state >= mtie_threshold){
            *setup_time = i+1;
            break;
        }
    }

    free(max_mat);
    free(min_mat);
    return steady_state;
}

/* End MTIE */

/* Settling-Time */
/* returns the last index i, where the error[i] was more than 10%
 * lower or higher than the average error.
 */
int settling_time(double *error){
    double avg = 0, min, max;
    int i;
    int settling = 0;

    for(i = 0; i < NUMBER_TRACE_MESSAGES; i++){
        avg += error[i];
    }
    avg = avg/(NUMBER_TRACE_MESSAGES);
    min = 1.1*avg;
    max = 0.9*avg;

    for(i = NUMBER_TRACE_MESSAGES - 1; i >= 0; i--){
        if(error[i]>max || error[i]<min){
            settling = i;
            exit;
        }
    }
    return settling;
}
/* End Settling-Time */

/* Auxillary Functions */
/* prints the whole population to file including parameters*/
void output(char * file){
    int i, j, index, size;
    individual * out;
    FILE * fp;
    index = get_first();
    out = get_individual(index);
    fp = fopen(file, "w");
```

```
    assert(fp != NULL);
    fprintf(fp, "alpha: %d, mu: %d, lambda: %d\t", alpha, mu, lambda);
    switch (ALGORITHM) {
        case 0:
            fprintf(fp, "LSDC6\n");
            fprintf(fp, "ID\tOBJ VAL 1\tOBJ VAL 2\tOBJ VAL 3\tDRIFT_VAR_INIT\t\
                 DRIFT_VAR_FACTOR\tDRIFT_VAR_FINAL\tINIT_PHASE\t\
                 WINDOW_SIZE\tESTIM_ERROR\tDRIFT_MAX\n");
            break;
        case 1:
            fprintf(fp, "PLL2\n");
            fprintf(fp, "ID\tOBJ VAL 1\tOBJ VAL 2\tOBJ VAL 3\tLIMIT\tK_I\tK_P\n");
            break;
        case 2:
            fprintf(fp, "LLR_CONT\n");
            fprintf(fp, "ID\tOBJ VAL 1\tOBJ VAL 2\tOBJ VAL 3\tBUFFER_SIZE\n");
            break;
        case 3:
            fprintf(fp, "MPEG\n");
            fprintf(fp, "ID\tOBJ VAL 1\tOBJ VAL 2\tOBJ VAL 3\tIOTA\tGAMMA\n");
            break;
    }
    size = get_size();
    for (i = 1; i < size; i++){
        index = get_next(index);
        out = get_individual(index);
        fprintf(fp, "%d\t%f\t%f\t%f\t", index, get_objective_value(index, 0),
                get_objective_value(index, 2), get_objective_value(index, 1));
        for(j = 0; j < NUMBER_PARAMETERS; j++)
            fprintf(fp, "%.12E\t", out->parameters[j]);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

/* prints the whole population's objective values to file w/o parameters*/
void output2(char * file){
    int i,index,size;
    individual * out;
    FILE * fp;
    index = get_first();
    out = get_individual(index);
    fp = fopen(file, "w");
    assert(fp != NULL);
    fprintf(fp, "ID\tOBJ VAL 1\tOBJ VAL 2\tOBJ VAL 3\n");
    size = get_size();
    for (i = 1; i < size; i++){
        index = get_next(index);
        out = get_individual(index);
```

```
        fprintf(fp, "%d\t%f\t%f\t%f\n", index, get_objective_value(index, 0),
                get_objective_value(index, 2), get_objective_value(index, 1));
    }
    fclose(fp);
}

int read_local_parameters()
/* Reads local parameters from file. */
{
    FILE *fp;

    int result,i;
    char str[CFG_ENTRY_LENGTH];

    /* reading cfg file with common configurations for both parts */
    fp = fopen(paramfile, "r");
    assert(fp != NULL);

    fscanf(fp, "%s", str);
    strcmp(str, "MAX_GEN");
    fscanf(fp, "%d", &MAX_GEN);
    assert(MAX_GEN > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "NUMBER_TRACE_MESSAGES") == 0);
    fscanf(fp, "%d", &NUMBER_TRACE_MESSAGES);
    assert(NUMBER_TRACE_MESSAGES > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "NUMBER_TRACE_FILES") == 0);
    fscanf(fp, "%d", &NUMBER_TRACE_FILES);
    assert(NUMBER_TRACE_FILES > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "TRACE_FILE_NAME:") == 0);
    for(i=0; i < NUMBER_TRACE_FILES; i++){
        fscanf(fp, "%s", TRACE_FILE_NAME[i]);
    }

    fscanf(fp, "%s", str);
    assert(strcmp(str, "OUTFILE_NAME") == 0);
    fscanf(fp, "%s", OUTFILE_NAME);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "ALGORITHM") == 0);
    fscanf(fp, "%d", &ALGORITHM);
    assert(ALGORITHM >= 0);

    fscanf(fp, "%s", str);
```

```
assert(strcmp(str, "NUMBER_OPT_PARAMETERS") == 0);
fscanf(fp, "%d", &NUMBER_OPT_PARAMETERS);
assert(NUMBER_OPT_PARAMETERS > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "NUMBER_PARAMETERS") == 0);
fscanf(fp, "%d", &NUMBER_PARAMETERS);
assert(NUMBER_PARAMETERS > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "EVAL_TIME") == 0);
fscanf(fp, "%d", &EVAL_TIME);
assert(EVAL_TIME > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "MTIE_WINDOW") == 0);
fscanf(fp, "%f", &MTIE_WINDOW);
assert(MTIE_WINDOW > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "SAMPLES_INT") == 0);
fscanf(fp, "%f", &SAMPLES_INT);
assert(SAMPLES_INT > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "MUTATION_FACTOR") == 0);
fscanf(fp, "%f", &MUTATION_FACTOR);
assert(MUTATION_FACTOR > 0);

result = fscanf(fp, "%s", str);
assert(strcmp(str, "MTIE_SETUP") == 0);
result = fscanf(fp, "%d", &MTIE_SETUP);
assert(MTIE_SETUP > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "OPT_JITTER") == 0);
fscanf(fp, "%d", &OPT_JITTER);
assert(OPT_JITTER >= 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "OPT_TVAR") == 0);
fscanf(fp, "%d", &OPT_TVAR);
assert(OPT_TVAR >= 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "OPT_MTIE") == 0);
fscanf(fp, "%d", &OPT_MTIE);
assert(OPT_MTIE >= 0);
```

```
    PRINT_ERROR = 0;

    fclose(fp);
    return (0);
}

/* Reads tracefile into memory*/
int read_tracefile(char *file_name, unsigned int *local_time,
                   unsigned int *remote_time, unsigned int *reference_time)
{
    int j;
    FILE *fp;
    int result;

    fp = fopen(file_name, "r");
    assert(fp != NULL);

    assert(local_time != NULL);
    assert(remote_time != NULL);
    assert(reference_time != NULL);

    for(j = 0; j < NUMBER_TRACE_MESSAGES; j++)
    {
        /* reading times fscanf() returns EOF if reading fails.*/
        result = fscanf(fp, "%u", &local_time[j]);
        result = fscanf(fp, "%u", &remote_time[j]);
        result = fscanf(fp, "%u", &reference_time[j]);

        if (result == EOF){
            fclose(fp);
            NUMBER_TRACE_MESSAGES = j;
            printf("NUMBER_TRACE_MESSAGES at most: %d\n", NUMBER_TRACE_MESSAGES);
            exit(1); /* signalling that not enough Messages in Tracefile*/
        }
    }
    fclose(fp);
    return (0);
}

void output_error(char *filename, double *error){
    FILE *fp;
    int i;
    fp = fopen(filename, "w");
    assert(fp != NULL);
    for(i = 0; i < NUMBER_TRACE_MESSAGES; i++){
        fprintf(fp, "%f\n", error[i]);
    }
}
```

```
void output_error2(char *filename, unsigned int *error){
    FILE *fp;
    int i;
    fp = fopen(filename, "w");
    assert(fp != NULL);
    for(i = 0; i < NUMBER_TRACE_MESSAGES; i++){
        fprintf(fp, "%u\n", error[i]);
    }
}
/* Auxillary Functions */
```

E.2 variator_user.h

```
/*==============================================================
 * variator_user.h
 *
 * Implementation of the simulation for time synchrinisation
 * algorithms with evolutionary methods.
 *
 * 2003/2004: Christian Hitz, Stefan Schuler
 *
 * Framework provided by TIK
 *============================================================*/
/*==============================================================
  PISA  (www.tik.ee.ethz.ch/pisa/)
  ============================================================
  Computer Engineering (TIK)
  ETH Zurich
  ============================================================
  PISALIB
  Pisa basic functionality that have to be implemented by the user
  Header file.
  file: variator_user.h
  author: Fabian Landis, flandis@ee.ethz.ch
  last change: 30.03.03
  by Stefan Bleuler, bleuler@tik.ee.ethz.ch
  ============================================================
*/

#ifndef VARIATOR_USER_H
#define VARIATOR_USER_H

/* maximal length of filenames */
#define FILE_NAME_LENGTH 128  /**** change the value if you like */

/* maximal length of entries in local cfg file */
#define CFG_NAME_LENGTH 128   /**** change the value if you like */
#define CFG_ENTRY_LENGTH 128
```

```
/* file to log to, defined in variator_user.c */
extern char *log_file;

/* file with local parameters */
char paramfile[FILE_NAME_LENGTH];

#define MAX_NUMBER_TRACE_FILES 20
#define MAX_NUMBER_PARAMETERS 7

int NUMBER_TRACE_MESSAGES;
char TRACE_FILE_NAME[MAX_NUMBER_TRACE_FILES][FILE_NAME_LENGTH];
int NUMBER_TRACE_FILES;
char OUTFILE_NAME[FILE_NAME_LENGTH];
int ALGORITHM;
int NUMBER_PARAMETERS;
int NUMBER_OPT_PARAMETERS;
int EVAL_TIME;
float MTIE_WINDOW;
float SAMPLES_INT;
int MTIE_SETUP;
float MUTATION_FACTOR;
int MAX_GEN, gen;
int OPT_JITTER;
int OPT_TVAR;
int OPT_MTIE;
int PRINT_ERROR;

unsigned int *local_time;
unsigned int *remote_time;
unsigned int *reference_time;

struct individual_t
{
    /* add your members here and make sure you initialize them
       correctly in create_individual */

    /* in variator.h a typedef is used to the define the type
       individual based on the individual_t struct defined here. */

    /* Parameter des Algorithmus */
    double parameters[MAX_NUMBER_PARAMETERS];

    /* Optimierungsvariablen */
    double jitter[MAX_NUMBER_TRACE_FILES]; /* objective value 0 */
    double tvar[MAX_NUMBER_TRACE_FILES];
    double mtie[MAX_NUMBER_TRACE_FILES];
    int setup_time[MAX_NUMBER_TRACE_FILES];
    int setup_time_j[MAX_NUMBER_TRACE_FILES];
};
```

```
/*------------------| functions for individual struct |---------------*/

void free_individual(individual *ind);
/* Frees the memory for one indiviual.

   post: memory for ind is freed
*/

double get_objective_value(int identity, int i);
/* Gets objective value of an individual.

   pre: 0 <= i <= dim - 1 (dim is the number of objectives)

   post: Return value == the objective value number 'i' in individual '*ind'.
         If no individual with ID 'identity' return value == -1.
*/

/*------------------------| statemachine |---------------------------*/
int state0();
/* Do what needs to be done in state 0.

   pre: The global variable 'paramfile' contains the name of the
        parameter file specified on the commandline.
        The global variable 'alpha' contains the number of indiviuals
        you need to generate for the initial population.

   post: Optionally read parameter specific for the module.
         Optionally do some initialization.
         Initial population created.
         Information about initial population written to the ini file
         using write_ini().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int state2();
/* Do what needs to be done in state 2.

   pre: The global variable 'mu' contains the number of indiviuals
        you need to read using 'read_sel()'.
        The global variable 'lambda' contains the number of individuals
        you need to create by variation of the individuals specified the
        'sel' file.

   post: Optionally call read_arc() in order to delete old uncessary
         individuals from the global population.
         read_sel() called
```

```
         'lambda' children generated from the 'mu' parents
         Children added to the global population using add_individual().
         Information about children written to the 'var' file using
         write_var().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int state4();
/* Do what needs to be done in state 4.

   pre: State 4 means the variator has to terminate.

   post: Free all memory.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int state7();
/* Do what needs to be done in state 7.

   pre: State 7 means that the selector has just terminated.

   post: You probably don't need to do anything, just return 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int state8();
/* Do what needs to be done in state 8.

   pre: State 8 means that the variator needs to reset and get ready to
        start again in state 0.

   post: Get ready to start again in state 0, this includes:
         Free all memory.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int state11();
/* Do what needs to be done in state 11.

   pre: State 11 means that the selector has just reset and is ready
        to start again in state 1.
```

```
   post: You probably don't need to do anything, just return 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int is_finished();
/* Tests if ending criterion of your algorithm applies.

   post: return value == 1 if optimization should stop
         return value == 0 if optimization should continue
*/
void init_population_lsdc(int * initial_population);
void init_population_pll(int * initial_population);
void init_population_llr(int * initial_population);
void init_population_mpeg(int * initial_population);
void evaluate(int *parent_identities, int *offspring_identities);
int lsdc6(int step, double *parameters, unsigned int *local_time,
          unsigned int *remote_time, double *ts, double *drift);
int llr_cont(double *parameters, unsigned int *local_time,
             unsigned int *remote_time, double *t, double *rho);
int mpeg_grad(double *parameters, unsigned int *local_time,
              unsigned int *remote_time, double *t, double *rho);
int pll2(double *parameters, unsigned int *local_time,
         unsigned int *remote_time, double *t, double *rho);
void compute_error(double *error, double* synchronized_time,
                   unsigned int* reference_time);
double jitter(unsigned int *reference_time, double* error,
              int* setup_time, int eval_time);
double mtieslow(double *error, int window, int samples, int setup);
int read_tracefile(char *file_name, unsigned int *local_time,
                   unsigned int *remote_time, unsigned int *reference_time);
double max_array(double *array, int i, int j);
double min_array(double *array, int i, int j);
int read_local_parameters();
void calculate_ov(int ind);
int recombine_weighted(individual * ind1, individual * ind2);
void recombine_crossover(individual * ind1, individual * ind2,
                         int * id1, int * id2);
int mutate(individual * ind);
void output(char *file);
void output2(char *file);
double tvar(double *error, int window, int samples, int setup);
double tvarslow(double *error, int window, int samples, int setup);
double mtie(int log2window, double *error, int* setup_time);
int settling_time(double *error);
void output_error(char *filename, double *error);
void output_error2(char *filename, unsigned int *error);
```

```
double mtie_average(individual* temp_ind);
double mtie_average2(individual* temp_ind);
double setup_time_average(individual* temp_ind);
double jitter_average(individual* temp_ind);
double jitter_average2(individual* temp_ind);
double setup_time_j_average(individual* temp_ind);
double worse(individual* temp_ind);

#endif /* VARIATOR_USER.H */
```

E.3 evaluate_parameter.c

```
/*============================================================
 * variator_user.c
 *
 * calculate Objective values from a given set of parameters.
 *
 * main2() wird vom main() in variator.c aufgerufen, falls der
 * Name des Binary "eval_param" enth\"alt.
 *
 * 2003/2004: Christian Hitz, Stefan Schuler
 *
 * Framework provided by TIK
 *============================================================*/

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <string.h>
#include <time.h>

#include "evaluate_parameter.h"

/* Auxillary Functions */
/* prints the whole population to file including parameters*/

int read_local_parameters_eval(individual *ind, char* param)
/* Reads local parameters from file. */
{
     FILE *fp;

     int result,i;
     char str[CFG_ENTRY_LENGTH];

     float a;

     /* reading cfg file with common configurations for both parts */
     fp = fopen(param, "r");
```

```
assert(fp != NULL);

fscanf(fp, "%s", str);
strcmp(str, "MAX_GEN");
fscanf(fp, "%d", &MAX_GEN);
assert(MAX_GEN > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "NUMBER_TRACE_MESSAGES") == 0);
fscanf(fp, "%d", &NUMBER_TRACE_MESSAGES);
assert(NUMBER_TRACE_MESSAGES > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "NUMBER_TRACE_FILES") == 0);
fscanf(fp, "%d", &NUMBER_TRACE_FILES);
assert(NUMBER_TRACE_FILES > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "TRACE_FILE_NAME:") == 0);
for(i=0; i < NUMBER_TRACE_FILES; i++){
     fscanf(fp, "%s", TRACE_FILE_NAME[i]);
}

fscanf(fp, "%s", str);
assert(strcmp(str, "OUTFILE_NAME") == 0);
fscanf(fp, "%s", OUTFILE_NAME);

fscanf(fp, "%s", str);
assert(strcmp(str, "ALGORITHM") == 0);
fscanf(fp, "%d", &ALGORITHM);
assert(ALGORITHM >= 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "NUMBER_OPT_PARAMETERS") == 0);
fscanf(fp, "%d", &NUMBER_OPT_PARAMETERS);
assert(NUMBER_OPT_PARAMETERS > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "NUMBER_PARAMETERS") == 0);
fscanf(fp, "%d", &NUMBER_PARAMETERS);
assert(NUMBER_PARAMETERS > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "EVAL_TIME") == 0);
fscanf(fp, "%d", &EVAL_TIME);
assert(EVAL_TIME > 0);

fscanf(fp, "%s", str);
assert(strcmp(str, "MTIE_WINDOW") == 0);
```

```
    fscanf(fp, "%f", &MTIE_WINDOW);
    assert(MTIE_WINDOW > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "SAMPLES_INT") == 0);
    fscanf(fp, "%f", &SAMPLES_INT);
    assert(SAMPLES_INT > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "MUTATION_FACTOR") == 0);
    fscanf(fp, "%f", &MUTATION_FACTOR);
    assert(MUTATION_FACTOR > 0);

    result = fscanf(fp, "%s", str);
    assert(strcmp(str, "MTIE_SETUP") == 0);
    result = fscanf(fp, "%d", &MTIE_SETUP);
    assert(MTIE_SETUP > 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "OPT_JITTER") == 0);
    fscanf(fp, "%d", &OPT_JITTER);
    assert(OPT_JITTER >= 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "OPT_TVAR") == 0);
    fscanf(fp, "%d", &OPT_TVAR);
    assert(OPT_TVAR >= 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "OPT_MTIE") == 0);
    fscanf(fp, "%d", &OPT_MTIE);
    assert(OPT_MTIE >= 0);

    fscanf(fp, "%s", str);
    assert(strcmp(str, "PRINT_ERROR") == 0);
    fscanf(fp, "%d", &PRINT_ERROR);
    assert(PRINT_ERROR >= 0);

    for(i = 0; i < NUMBER_PARAMETERS; i++){
        result = fscanf(fp, "%f", &a);
        ind->parameters[i] = a;
        assert(result != EOF); /* no EOF, correctly read */
        assert(ind->parameters[i] >= 0);
    }

    fclose(fp);
    return (0);
}
```

```
int main2(int argc, char *argv[]){
    int i, indiv;
    extern int dimension;
    individual *ind;
    dimension = 3;

    ind = (individual*)malloc(sizeof(individual));
    indiv = add_individual(ind);

    read_local_parameters_eval(ind, argv[1]);
    switch (ALGORITHM) {
        case 0:
            printf("Algorithm: LSDC6\n");
            break;
        case 1:
            printf("Algorithm: PLL2\n");
            break;
        case 2:
            printf("Algorithm: LLR_CONT\n");
            break;
        case 3:
            printf("Algorithm: MPEG\n");
            break;
    }
    local_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                 NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(local_time != NULL);
    remote_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                  NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(remote_time != NULL);
    reference_time = (unsigned int*)malloc(NUMBER_TRACE_FILES*
                     NUMBER_TRACE_MESSAGES*sizeof(unsigned int));
    assert(reference_time != NULL);

    for(i = 0; i < NUMBER_TRACE_FILES; i++){
        read_tracefile(TRACE_FILE_NAME[i], &local_time[i*NUMBER_TRACE_MESSAGES],
                       &remote_time[i*NUMBER_TRACE_MESSAGES],
                       &reference_time[i*NUMBER_TRACE_MESSAGES]);
    }

    calculate_ov(indiv);
    printf("Objective Value 1: %f\nObjective Value 2: %f\n\
Objective Value 3: %f\n", get_objective_value(indiv, 0),
            get_objective_value(indiv, 1), get_objective_value(indiv, 2));

    return(0);
}
```

E.4 evaluate_parameter.h

```
#ifndef EVALUATE_PARAMETER_H
#define EVALUATE_PARAMETER_H

#include "variator.h"
#include "variator_user.h"

int read_local_parameters(individual *ind, char *param);
int main2(int argc, char* argv[]);

#endif /* EVALUATE_PARAMETER.H */
```

E.5 variator_param.txt

```
MAX_GEN 400
NUMBER_TRACE_MESSAGES 8788
NUMBER_TRACE_FILES 20
TRACE_FILE_NAME:
../auswertung/rAdHoc01/normalized/[01]AdGrklein200.txt
../auswertung/rAdHoc02/normalized/[02]AdGrklein200.txt
../auswertung/rAdHoc03/normalized/[03]AdGrklein200.txt
../auswertung/rAdHoc04/normalized/[04]AdGrklein200.txt
../auswertung/rAdHoc05/normalized/[05]AdGrklein200.txt
../auswertung/rAdHoc06/normalized/[06]AdGrklein200.txt
../auswertung/rAdHoc07/normalized/[07]AdGrklein200.txt
../auswertung/rAdHoc08/normalized/[08]AdGrklein200.txt
../auswertung/rAdHoc09/normalized/[09]AdGrklein200.txt
../auswertung/rAdHoc10/normalized/[10]AdGrklein200.txt
../auswertung/rAdHoc01/normalized/[01]AdGrgross200.txt
../auswertung/rAdHoc02/normalized/[02]AdGrgross200.txt
../auswertung/rAdHoc03/normalized/[03]AdGrgross200.txt
../auswertung/rAdHoc04/normalized/[04]AdGrgross200.txt
../auswertung/rAdHoc05/normalized/[05]AdGrgross200.txt
../auswertung/rAdHoc06/normalized/[06]AdGrgross200.txt
../auswertung/rAdHoc07/normalized/[07]AdGrgross200.txt
../auswertung/rAdHoc08/normalized/[08]AdGrgross200.txt
../auswertung/rAdHoc09/normalized/[09]AdGrgross200.txt
../auswertung/rAdHoc10/normalized/[10]AdGrgross200.txt
OUTFILE_NAME AdGr4/AdGr/AdGr
ALGORITHM 0
NUMBER_OPT_PARAMETERS 6
NUMBER_PARAMETERS 7
EVAL_TIME 10
MTIE_WINDOW 8
SAMPLES_INT 0.2
MUTATION_FACTOR 0.3
MTIE_SETUP 10
OPT_JITTER 0
```

```
OPT_TVAR 0
OPT_MTIE 1
PRINT_ERROR 0
```

Anhang F

MATLAB Skripte

F.1 analyse_parameter.m

```
function analyse_parameter(file)
%ANALYSE_PARAMETER  Plots objective values an corresponding parameters
%  file: "*.in.txt" or "*.out.txt" from the PISA simulation
%
data = dlmread(file,' ', 2, 0);
% data = data(:,[1:4 6:12]);
pm = data(:,1:4);
pp = dominatePts(pm');
pp = pp';
pareto = data(pp(:,1),:);
pareto = sortrows(pareto,[2 4]);
paretoshow = pareto(:,1:11);

label = [
    'MTIE\_klein       ';
    'Setup Time        ';
    'MTIE\_gross       ';
    'DRIFT\_VAR\_INIT  ';
    'DRIFT\_VAR\_FACTOR';
    'DRIFT\_VAR\_FINAL ';
    'INIT\_PHASE       ';
    'WINDOW\_SIZE      ';
    'ESTIM\_ERROR      '];

size = 9;

for i=1:size
    subplot(size,1,i);
    if i == 7
        plot(floor(pareto(:,i+1)));
    else
        plot(pareto(:,i+1));
```

```
    end
    ylabel(label(i,:));
end
```

F.2 dominatePts.c

```
/* =================================================================
 * dominatePts.c
 *
 * takes Matrix and returns Paretoelements
 *
 * (on avarage 7 times faster than the implementation in MATLAB)
 *
 * 6.1.2004 Christian Hitz
 */

#include "mex.h"

int dominates(int number_of_dims, double *a, double *b);

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]){
    /* Declare variables. */
    int elements, j, i, n, m, cmplx, size;
    int elementSize, dominated, numberRows, rowLength, rowSize;
    int nnz = 0, count = 0;
    double *pr, *pi, *pind, *pind2;
    const int  *dim_array;
    mxArray *out;

    /* Check for proper number of input and output arguments. */
    if (nrhs != 1) {
        mexErrMsgTxt("One input argument required.");
    }
    if (nlhs > 1) {
        mexErrMsgTxt("Too many output arguments.");
    }

    /* Check data type of input argument. */
    if (!(mxIsDouble(prhs[0]))) {
        mexErrMsgTxt("Input array must be of type double.");
    }

    /* Get the number of elements in the input argument. */
    elements = mxGetNumberOfElements(prhs[0]);

    /* Get the data. */
    pr = (double *)mxGetPr(prhs[0]);
    pi = (double *)mxGetPi(prhs[0]);
    cmplx = ((pi == NULL) ? 0 : 1);
```

```
    dim_array = mxGetDimensions(prhs[0]);

    /* Get the number of dimensions in the input argument.
       Allocate the space for the return argument */
    numberRows = dim_array[1];
    rowLength = dim_array[0];
    out = mxCreateDoubleMatrix(rowLength, numberRows, mxREAL);
    pind = mxGetPr(out);
    elementSize = mxGetElementSize(out);
    rowSize = rowLength*elementSize;

    j = 0;
    /* Fill in the indices to return to MATLAB.*/
    for (m = 0; m < numberRows; m++) {
        dominated = 0;
        for(n = 0; n < numberRows; n++){
            if((dominated = dominates(rowLength, &pr[n*rowLength],
                                      &pr[m*rowLength]))){
                break;
            }
        }
        if(dominated == 0){
            for(i = 0; i < rowLength; i++){
                pind[j*rowLength+i] = pr[m*rowLength+i];
            }
            j++;
        }
    }

    plhs[0] = mxCreateDoubleMatrix(rowLength, j, mxREAL);
    pind2 = mxGetPr(plhs[0]);
    memcpy(pind2, pind, j*rowSize);
    mxDestroyArray(out);
    return;
}

int dominates(int rowLength, double *a, double *b){
    int equal = 1;
    int worse = 0;
    int i;

    for(i = 1; i < rowLength; i++){
        worse = worse || (a[i] > b[i]);
        equal = equal && (a[i] == b[i]);
    }

    return !equal && !worse;
}
```

F.3 paretoplot3d.m

```
function paretoplot3d(file, pareto, setup_threshold)
%PARETOPLOT3D Plots paretopoints in 3 dimensions and all their 3
%projections
%  file: generations file from the PISA simulation
%  pareto: 1: search paretopoints; 0: plot all points
%  setup_threshold: plot all point with setup_time < setup_threshold
%                   as red circles

data = dlmread(file,' ', 1, 0);
if(pareto == 1)
    data = data';
    non_dom2 = dominatePts(data);
    non_dom2 = non_dom2';
else
    non_dom2 = data;
end
% close
subplot(2,2,3)
plot3(non_dom2(:,2), non_dom2(:,3), non_dom2(:,4), '*');
grid on
xlabel('average MTIE kleine Last')
ylabel('average Setup Time [s]')
zlabel('average MTIE grosse Last')
subplot(2,2,4)
plot(non_dom2(:,2), non_dom2(:,4), '*');
xlabel('average MTIE kleine Last')
ylabel('average MTIE grosse Last')
subplot(2,2,2)
plot(non_dom2(:,3), non_dom2(:,4), '*');
xlabel('average Setup Time [s]')
ylabel('average MTIE grosse Last')
subplot(2,2,1)
plot(non_dom2(:,2), non_dom2(:,3), '*');
xlabel('average MTIE kleine Last')
ylabel('average Setup Time [s]')

length = size (non_dom2);
%hold on;
j=1;
if setup_threshold > 0
    for i=1:length(1,1)
        if (non_dom2(i,3) > setup_threshold)
            non_dom3(j,:) = non_dom2(i,:);
            j = j+1;
        end
    end
    non_dom3;
```

```
    subplot(2,2,4)
    hold on;
    plot(non_dom3(:,2), non_dom3(:,4),'r*');
    subplot(2,2,2)
    hold on;
    plot(non_dom3(:,3), non_dom3(:,4),'r*');
    subplot(2,2,1)
    hold on;
    plot(non_dom3(:,2), non_dom3(:,3),'r*');
end
```

F.4 pareto_movie.m

```
function [movie,v] = pareto_movie(filenamebase, outfilebase,...
    generations, one, two, v)
%PARETO_MOVIE Make a movie of paretoplots
%  filenamebase: infiles
%  outfilebase: outfiles (only written, when activated in pareto_plot()
%  generations: number of generations
%  one: objective value 1
%  two: objective value 2
%  v: axis ([xmin xmax ymin ymax])

for i = 0:generations-1
    i
    h = pareto_plot([filenamebase '.' num2str(i) '.txt'],...
        [outfilebase '.' num2str(i) '.out.txt'], one, two);
    axis(v);
    movie(i+1) = getframe(h);
    close;
end
```

F.5 pareto_plot.m

```
function h = pareto_plot(filename, outputfile, one, two)
%PARETO_PLOT
% Plots the nondominated front of set of 2-dimensional points. Objective
% values are minimized.
%
% Usage:
%   filename: infile from PISA simulation
%   outfile: filename for output
%   one: first objective value
%   two: second objective value
%
% Pre: 'file.txt' contains one line of titles.
%
%      All other lines in between have three columns:
```

```
%      1. ID of a solution (ID is an integer)
%      2. first objective
%      3. second objective
%
%      These columns are separated by a space.
%
% Post: The nondominated individuals are plotted.
%
%       The IDs and objective values of the nondominated individuals are
%       sorted according to ascending values of the first objective and are
%       written to 'outputfile'.
%
% Stefan Bleuler, bleuler@tik.ee.ethz.ch
% TIK, ETH Zurich
%
% 21.11.03
%
% Christian Hitz, Stefan Schuler:
% Changed to C implementation of nondominated individuals search


data = dlmread(filename,' ', 1, 0); % 1 means ignoring the first line

% extracting the nondominated solutions
non_dom = [];
one = one + 1;
two = two + 1;
step = two - one;

data2 = data';
non_dom2 = dominatePts(data2); %call C-Function
non_dom2 = non_dom2';
non_dom = [non_dom2(:,1) non_dom2(:,one:step:two)];

% write nondominated to 'outputfile'

%non_dom = [data(:,1) data(:,one:step:two)];

non_dom = sortrows(non_dom, 2);

dlmwrite(outputfile, non_dom, ' ')
grid on
% plotting

h = figure;
plot(non_dom(:, 2), non_dom(:, 3), '*');
xlabel('first objective');
ylabel('second objective');
```

```
% labeling

% hold on;
% e_x = max(non_dom(:,2)) / 100; % an epsilon to add to the ...
% e_y = max(non_dom(:,2)) / 100; % position of the text
% for(k = 1:size(non_dom, 1))
%     text(non_dom(k,2) + e_x, non_dom(k,3) + e_y, num2str(non_dom(k,1)));
% end
% saveas(h,[outputfile '.fig']);

% write nondominated to 'outputfile'
% dlmwrite(outputfile, non_dom, ' ')
```

F.6 parsemain.m

```
function parsemain()
%PARSEMAIN
%  Normalisiert alle Tracefiles im aktuellen Ordner
%  Aus ref und sync resultiert ein file
%  local_time, remote_time, reference_time

mkdir normalized;
ls -X -1 *ref*.txt;
reference_files = ans;
ls -X -1 *sync*.txt;
timestamp_files = ans;
nexts = 1;
nextr = 1;
[b,count] = sscanf(timestamp_files,'%s');
for i=1:count
    [timestamp_filename,b,c,nextsNew] =...
        sscanf(timestamp_files(nexts:length(timestamp_files)),'%s',1);
    [reference_filename,b,c,nextrNew] =...
        sscanf(reference_files(nextr:length(reference_files)),'%s',1);
    timestamp_filename
    reference_filename
    nexts = nexts + nextsNew-1;
    nextr = nextr + nextrNew-1;
    new_filename = ['normalized/' strrep(timestamp_filename,'sync','')]
    [local_time,remote_time,reference_time] =...
        parse_tracefiles_write(timestamp_filename,reference_filename,new_filename);
end
```

F.7 analyse_trace.m

```
function analyse_trace(trace_filename,von,bis)
%ANALYSE_TRACE  Plots the timedelays between adjacent timestamps
```

```
%  trace_filename: normalized tracefile
%  von: first timestamp
%  bis: last timestamp

% parse timestamp file
fid=fopen(trace_filename);
if ((fid == -1))
    disp(['Could not open ' timestamp_filename]);
    return;
end

a = fscanf(fid,'%u%u%u',[3 inf]);

fclose(fid);
a = a';
e = length(a);
e1 = e-1;
b = a(2:e,1)-a(1:e1,1);

plot(b(von:bis),'.b');
ylabel('Abstand der Tracemessages');
xlabel('Samples');
```

Literaturverzeichnis

[1] *PISA*. http://www.tik.ee.ethz.ch/pisa.

[2] *VideoLAN (VLC)*. http://www.videolan.org.

[3] Stefan Bleuler. PISA, a platform and programming language independent interface for search alorithms. Technical report, Institute TIK, Department of Information Technology and Electrical Engeneering, ETH Zurich, 2003.

[4] Philipp Blum and Georg Dickmann. A benchmark for clock synchronisation algorithms in wired and wireless local area networks. Unpublished, 2003.

[5] Philipp Blum and Georg Dickmann. Precise delay measurements in wired and wireless local area networks. Technical Report 175, Institute TIK, Department of Information Technology and Electrical Engeneering, ETH Zurich, 2003.

[6] Philipp Blum and Lothar Thiele. Precise and Low-Jitter Wireless Time Synchronization. Technical Report 187, Institute TIK, Department of Information Technology and Electrical Engeneering, ETH Zurich, 2003.

[7] Stefano Bregni. Fast Algorithms for TVAR and MTIE Computeration in Characterization of Network Synchronization Performance. In G. Antoniou, N. Mastorakis, and O. Panfilov, editors, *Advances in Signal Processing and Computer Technologies*. WSES Press, 2001.

[8] Eckart Zitzler. Studien- und Diplomarbeiten, Merkblatt für Studenten und Betreuer. Internes Memo, March 1998.