ComeTogether - an android app to meet new people

Christof Ochmann
University of Applied Sciences Zittau/Görlitz
02826 Görlitz, Germany
sichochm@hs-zigr.de

Ingo Körner
University of Applied Sciences Zittau/Görlitz
02826 Görlitz, Germany
siinkoer@hs-zigr.de

ABSTRACT

This work based on the document ”Come-Together-App”, which was created in Wirtschaftsinformatik II in a course of studies of tourism. The ideas of the project ”Come-Together-App” are realized as a prototype in ”ComeTogether - an android app to meet new people”. Both, frontend and backend are analysed, desgined and implemented.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Design, Performance, Theory

Keywords

Parallel evolutionary algorithms, island model, spatial structures, offspring populations, runtime analysis

1. INTRODUCTION

In this project a prototyp named ComeTogether will be created. He runs on smartphones with the android operating system. ComeTogether is a mixture of an eventcalendar and a platonic touch. ComeTogether is an app to establish social contacts. It is no flirt or dating app, but an app to meet immediately a whole group of new people simultaneously and have fun with them. The core of the application consists of an offer function and a search function. With offer you can offer events with a participating group of people. With search you can find a group of people which participate in the event you have searched. Both, users who offer and users who search signalise with an offer or a search, that they want actively meet new people.

2. DESIGN

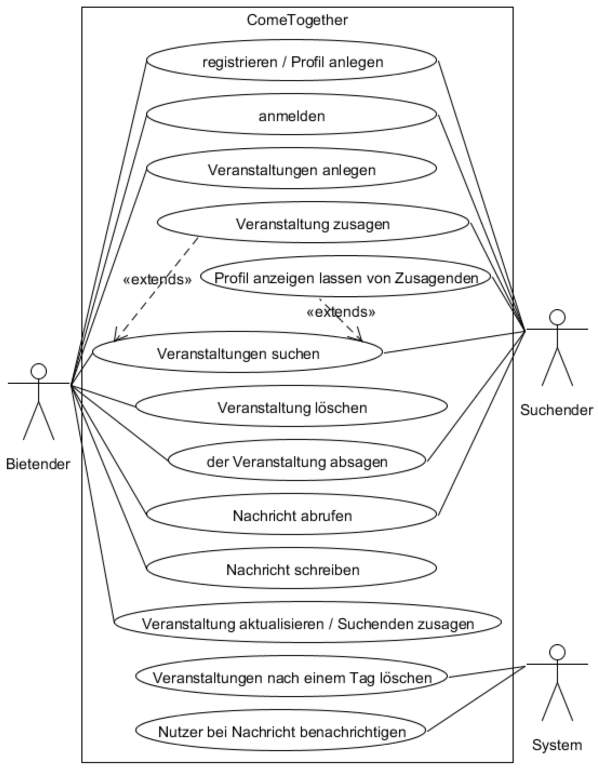
ComeTogether
registrieren / Profil anlegen
anmelden
Veranstaltungen anlegen
Veranstaltung zusagen
«extends»
Profil anzeigen lassen von Zusagenden
«extends»
Veranstaltungen suchen
Veranstaltung löschen
der Veranstaltung absagen
Nachricht abrufen
Nachricht schreiben
Veranstaltung aktualisieren / Suchenden zusagen
Veranstaltungen nach einem Tag löschen
Nutzer bei Nachricht benachrichtigen
Bietender
Suchender
System

Figure 1: use case diagram

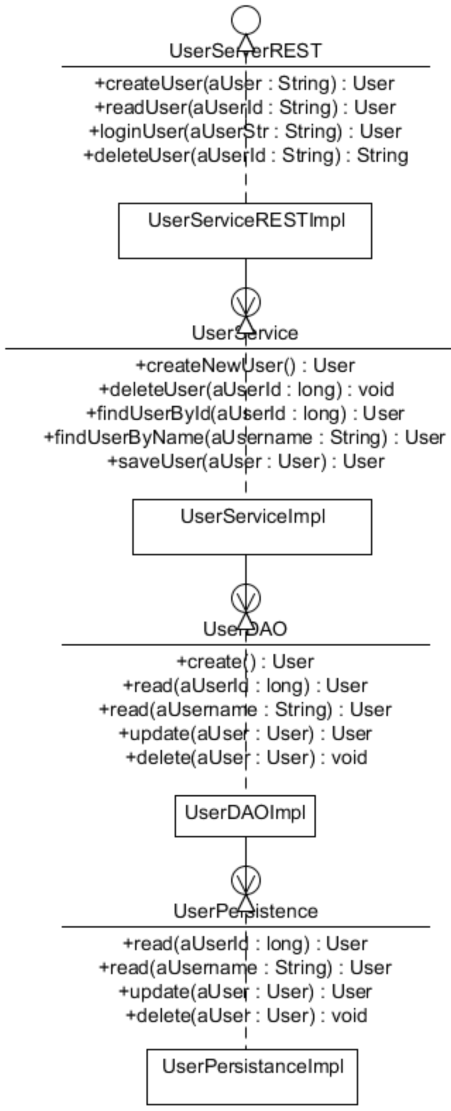
UserServerREST
+createUser(aUser : String) : User
+readUser(aUserId : String) : User
+loginUser(aUserStr : String) : User
+deleteUser(aUserId : String) : String
UserServiceRESTImpl
UserService
+createNewUser() : User
+deleteUser(aUserId : long) : void
+findUserById(aUserId : long) : User
+findUserByName(aUsername : String) : User
+saveUser(aUser : User) : User
UserServiceImpl
UserDAO
+create() : User
+read(aUserId : long) : User
+read(aUsername : String) : User
+update(aUser : User) : User
+delete(aUser : User) : void
UserDAOImpl
UserPersistence
+read(aUserId : long) : User
+read(aUsername : String) : User
+update(aUser : User) : User
+delete(aUser : User) : void
UserPersistanceImpl

Figure 2: desing class diagramm UserServiceREST

3. DATABASE

Parallelization is becoming a more and more important issue for solving difficult optimization problems [2]. Evolutionary Algorithms (EAs) are known to be one of the most successful strategies to find good solutions of hard optimization problems for which no efficient methods are known. For large-scale applications it is common to use parallel implementations, which have shown to be a successful strategy to speed up the computation in many cases [1, 16]. Especially due to current multi-core architectures, parallel evolutionary algorithms are highly relevant [3, 16].

A simple way of using parallelization is to use offspring populations, where offspring are created and evaluated on different processors. Island models use parallelization on a higher level. Subpopulations, called islands, evolve independently for some time and periodically exchange individuals in a process called migration. Islands are usually spatially structured, with communication taking place on a communication topology connecting the islands. Despite many widespread applications and a long history of such parallel EAs, the theory of these algorithms is not well developed and more fundamental research is needed [15].

The most obvious goal of parallelization is to obtain a good speedup, defined as the quotient of the total optimization time such as the total number of function evaluations in an EA and the parallel optimization time such as the number of generations. (Asymptotic) linear speedups are achieved when the parallel running time is (asymptotically) the total optimization time divided by the number of processors. Recently, Lässig and Sudholt [8, 9] presented a constructed function where an exponential speedup could be shown. In [10] they provided general methods for analyzing expected parallel optimization times—and hence speedups—by adapting the well-known fitness level method. This was applied to island models with different migration topologies [10] such as ring graphs, torus graphs, and complete graphs for basic test functions such as ....

An open question is how island models perform in more practical settings such as combinatorial optimization where many analyses of (non-parallel) EAs have been obtained [13]. Therefore, we present such an analysis, considering speedups by parallelization for different combinatorial problems: sorting as an optimization problem [14] (Section 6), the single-source shortest path problem [4, 7] (Section 7), and the Eulerian cycle problem with different genotype representations and operators [5, 6, 12] (Section 8).

The goal is to investigate the possible speedups, identify settings where parallelization is most useful, and give answers to many design questions that come with island models such as how to choose the number of islands or the frequency of migration. Although these problems are easy from the perspective of computational complexity, our investigations give insight into how parallel EAs deal with characteristics of different problem classes, such as plateaus of equal fitness. This leads to interesting insights and paves the way for further investigations, including NP-hard problems.

4. PRELIMINARIES

The idea of an island model is to have separate subpopulations evolving independently for some time. ....

5. PREVIOUS WORK

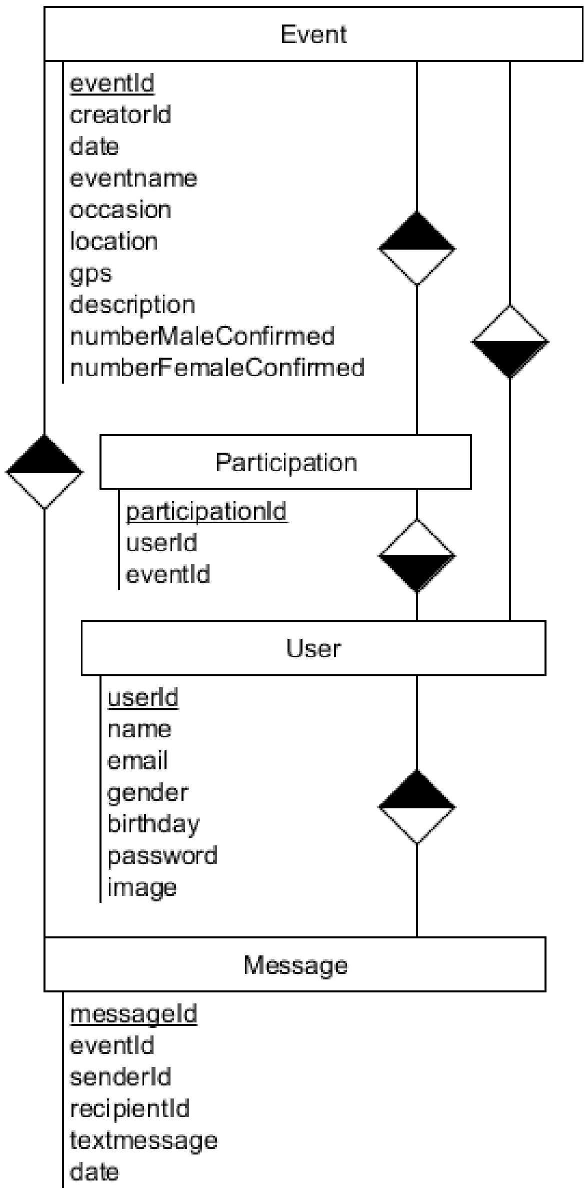

Figure 3: eer-diagram

Lässig and Sudholt [10, 11] presented general bounds for parallel EAs using the *fitness-level method* or *method of f-based partitions*. The idea is to divide the search space into sets $A_1, \dots, A_m$ strictly ordered w. r. t. fitness: $A_1 <_f A_2 <_f \cdots <_f A_m$ where $A <_f B$ iff $f(a) < f(b)$ for every $a \in A, b \in B$. In addition, A_m contains only global optima.

We say that a population-based search algorithm (this includes populations of size 1) is in A_i or on level i if the current best individual in the population is in A_i. Elitist algorithms (i. e. algorithms where the best solution in the population never worsens) can only increase the current level. The goal of the optimization is to reach A_m.

In the worst case an algorithm has to traverse all fitness levels $A_i, A_{i+1}, \dots, A_m$ if it starts in A_i. Now, if s_i is a lower bound on the probability of leaving A_i towards any higher fitness level in one generation, the expected time until this happens is at most the reciprocal, $1/s_i$. We then get that the optimization time is bounded by $\sum_{i=1}^{m-1} 1/s_i$.

The fitness-level method gives good bounds in case a search algorithm typically does not skip too many fitness levels and the probabilities of finding improvements from any set A_i are similar for all search points in A_i. It only yields crude upper bounds in case there are difficult fitness levels that are skipped with a very high probability.

The perspective used when arguing with fitness levels is very well suited for parallelization. The search for improvements is given by Bernoulli trials: in every generation there is a chance of finding an improvement. While waiting for improvements to be found, parallelizing these trials on different processors easily leads to good speedups. Lässig and Sudholt [10] have made this precise for island models that run elitist islands. If migration is used in every generation, information about the current best fitness level is propagated to neighbored islands. This increases the number of islands on the current best fitness level and thus the number of processors that search for better fitness levels in parallel.

The following theorem summarizes their results. The bounds for rings and torus graphs are refined towards bounds that hold for arbitrary numbers of islands; the theorems in [10] were restricted to large enough numbers. Our refined theorem allows for better insight into which numbers of islands lead to a linear speedup (asymptotically). A directed graph is strongly connected if for every two vertices u, v there is a directed path from u to v and vice versa.

Theorem 1. *Consider an island model with μ islands ...*

Proof. The third claim is a special case of Theorem 8 in [10] (results in [10] also accounted for stochastic migrations), allowing for a better constant in the factor $2/\mu$.

The first bound is a refinement of Theorem 3 in [10]. In the proof of this theorem it is shown that for every integer $k \le \mu$ the expected time until fitness level i is left is bounded by $k + \frac{2}{k} \cdot \frac{1}{s_i}$. Now, if $\mu \ge k := s_i^{-1/2}$ (ignoring rounding issues), the expected number of generations on fitness level i is bounded by $k + 2/k \cdot s_i^{-1} = s^{-1/2} + 2s^{-1/2}$. If $\mu < s_i^{-1/2}$, we get for $k := \mu$ an upper bound of $\mu + 2/\mu \cdot s_i^{-1} > s_i^{-1/2} + 2/\mu \cdot s_i^{-1}$. Together, this proves the claimed bound.

Likewise, the second bound refines Theorem 4 in [10] where it was proven that the expected time for leaving level i is at most $8\sqrt{k} + \frac{2}{k} \cdot \frac{1}{s_i}$. If $\mu \ge k := s_i^{-2/3}$, this gives $8s_i^{-1/3} + 2s_i^{-1/3} = 10s_i^{-1/3}$. Otherwise, $k := \mu$ yields a bound of $8\sqrt{\mu} + \frac{2}{\mu} \cdot \frac{1}{s_i} \le 8s_i^{-1/3} + \frac{2}{\mu} \cdot \frac{1}{s_i}$. $\square$

Assuming the fitness-level bound for the time $\sum_{i=1}^{m-1} \frac{1}{s_i}$ of a single island is asymptotically tight, all three bounds yield an asymptotic linear speedup in case the first summands are each of at most the same order as the second summand.

Besides the mentioned works, parallel models, including offspring populations, have also been considered in other theoretical studies of EAs in combinatorial optimization. ...

6. SORTING

...

7. SHORTEST PATHS

...

8. EULERIAN CYCLES

...

8.1 Edge Walks

...

8.2 Restricted Mutation Operators

...

8.3 Adjacency List Matchings

...

9. CONCLUSIONS

Considering speedups of island models has led to a surprising richness of results. For sorting linear speedups are possible, but only for $\mu = O(\log n)$ islands. The single-source shortest paths problem allows for linear speedups, the maximum number of islands depending on the topology and the mutation operator. For K_μ or offspring populations almost quadratic factors can be gained. For Eulerian cycles and the edge walk representation speedups on the instance G' vary grossly from exponential up to $\mu = O(\log m)$ for rare or no migrations to at most logarithmic speedups, if migration is used too frequently. This is due to the existence of plateaus of equal fitness. Again, results vary with the mutation operator and the representation. The most efficient representation, adjacency list matchings, turned out to be parallelizable efficiently in a straightforward way.

Acknowledgments

The authors would like to thank .....the German fast food industry for keeping us alive.

10. REFERENCES

[1] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

[2] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms.* Wiley-Interscience, 2005.

[3] E. Cantú-Paz and D. E. Goldberg. On the scalability of parallel genetic algorithms. *Evolutionary Computation*, 7(4):429–449, 1999.

[4] B. Doerr, E. Happ, and C. Klein. A tight analysis of the (1+1)-EA for the single source shortest path problem. In *Proc. of CEC '07*, pages 1890–1895. IEEE, 2007.

[5] B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation*, 15:401–410, 2007.

[6] B. Doerr and D. Johannsen. Adjacency list matchings—an ideal genotype for cycle covers. In *Proc. of GECCO '07*, pages 1203–1210. ACM, 2007.

[7] B. Doerr and D. Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Proc. of GECCO '10*, pages 759–766. ACM, 2010.

[8] J. Lässig and D. Sudholt. The benefit of migration in parallel evolutionary algorithms. In *Proc. of GECCO '10*, pages 1105–1112, 2010.

[9] J. Lässig and D. Sudholt. Experimental supplements to the theoretical analysis of migration in the island model. In *PPSN '10*, pages 224–233. Springer, 2010.

[10] J. Lässig and D. Sudholt. General scheme for analyzing running times of parallel evolutionary algorithms. In *PPSN '10*, pages 234–243. Springer, 2010.

[11] J. Lässig and D. Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proc. of FOGA '11*, 2011. To appear.

[12] F. Neumann. Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers & Operations Research*, 35(9):2750–2759, 2008.

[13] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity.* Springer, 2010.

[14] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004.

[15] Z. Skolicki and K. A. De Jong. The influence of migration sizes and intervals on island models. In *Proc. of GECCO '05*, pages 1295–1302. ACM, 2005.

[16] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time.* Springer, 2005.