سورت بهینه با الگوریتمهای ژنتیک

چکیده

رشد پیچیدگی پردازشهای مدرن تولید کد موثر مناسب را بصورت افزایشی سخت ساخته است. تولید کد به صورت دستی بسیار زمان صرف کننده می باشد اما ان بارها انتخاب میشوند به طوریکه کد تولید شده بوسیله تکنولوژی کامپایلرهای امروزی بارها اجرای پایین تری نسبت به بهترین کدهای اماده شده دستی دارند. یک نولید از استراتژی تولید کد انجام گرفته شده بوسیله سیستمهای شبیه ATLAS,FFTW و SPIRAL که جستجوی تجربی را برای پیدا کردن مقادیر پارامتر از کارایی را استفاده کرده است به طوریکه اندازه تیله و زمانبندی آموزش که تحویل انجام بهینه برای یک ماشین مخصوص میباشد. به هر حال این دیدگاه دارد به تنهایی به طور کامل ثابت میکند در کدهای علمی اجرا به داده ورودی وابسته نیست. در این مقاله ما مطالعه میکنیم تکنیکهای یادگیری ماشین را برای توسعه جستجوی ترتیبی برای تولیدی از روتینهای سورت کردن که اجرا در مشخصات ورودی و معماری از ماشین هدف وابسته است. ما ساختیم در مطالعه قبلی که یک الگوریتم سورت خالص در اغازی از محاسبات مثل تابعی از انحراف معیار انتخاب کنیم. دیدگاهی که بحث میکنیم در این مقاله الگوریتمهای ژنتیک و یک سیستم طبقه بندی با ساختار بصورت سلسله مراتبی ساخته شده الگوریتمهای سورت تلفیقی توانا از تبدیل کردن داده ورودی استفاده میکند. نتایج ما نشان میدهد که الگوریتمهای تولید شده با استفاده از دیدگاه ارائه شده در این مقاله هستند سرع و موثر در گرفتن داخل اکانت اثرات متقابل پیچیده ما بین معماری و مشخصات داده ورودی و کد نتیجه بسیار مهم بهتر از اجراعات سورت مرسوم و کد تولید شده با مطالعه اسان ما اجرا میکند. به ویژه روتینهای تولید شده با دیدگاه ما اجرا میکند بهتر از تمام کتابخانه های تجاری که ما ازمایش کردیم مثل IBM ESSL,INTEL MKL و C++ STL. بهترین الگوریتم ما دارد توانایی تولید ای در معدل 26٪ و 62٪ سریعتر از IBM ESSL در یک IBM PAWER 3 و IBM PAWER 4 بترتیب را دارد.

1 مقدمه

اگر چه تکنولوژی کامپایلر فوق العاده در پردازش خودکاراز بهینه سازی برنامه کامل شده است و بیشتر مداخلات انسانی هنوز هست برای تامین کد بسیار سریع لازم شده است. یک دلیل اینکه ناجوری از اجراعات کامپایلر وجود دارد. اینها کامپایلرهای بیهنه عالی برای بعضی پلاتفرمها هستند اما کامپایلرهای موجود برای بعضی پلاتفرمهای دیگر بسیاری خواسته ها را ترک میکنند. دومین دلیل و شاید بسیار مهم این هست که کامپایلرهای مرسوم که فاقد اطلاعات معنایی هستند و بنابراین محدود شده اند به قدرت دگرگونییا تغییر. یک دیدگاه ایجاد شده که دارد ثابت شده بکلی موثر در چیره شدن به هر دوی این محدودیتها استفاده نمودن تولید کننده های کتابخانه میباشد. این سیستمها استفاده معنایی در اطلاعات برای بکار بردن دگرگون سازی در تمام سطوح از تجرید مهیا میسازند. بیشتر تولیدات کتابخانه قدرتمند نیستند فقط بهینه ساز برنامه همان سیستمهای طراحی الگوریتم هستند.

ترجمه فا

ATLAS[21],PHiPAC[2],FFTW[7],SPIRAL[23] در طول بهترین تولید کننده های کتابخانه دانسته شده میباشند. ATLAS ,PHiPAC تولید میکنند روتینهای جبری خطی را و پردازش بهینه را در پیاده سازی از ضرب ماتریس در ماتریس فوکس میکنند. در مدت نصب مقادیر پارامتر از یک پیاده سازی ضرب یک ماتریس بطوریکه اندازه تیله و مقداری از حلقه باز شده که تحویل میدهد بهترین انجام معین کننده هویت استفاده جستجوی تجربی. این جستجو پردازش میشود با تولید کردن ورژنهای گوناگون از ضرب ماتریسی که تنها اختلاف دارند در مقدار پارامتر که هست شروع به جستجو. در تقریب جستجوی گسترده هست استفاده شده برای پیدا کردن بهترین مقادیر پارامتر. دو سیستم دیگر اشاره دارند روی SPIRAL,FFTW تولید میکنند کتابخانه های پردازش کننده تنها را. فضای جستجو در SPIRAL,FFTW هست همچنین بزرگتر برای جستجوی گسترده برای ممکن شدن. بنابراین این سیستمها جستجو میکنند با استفاده هیوریستیک مثل برنامه نویسی داینامیک [7,12] یا الگوریتمهای ژنتیک [19].

در این مقاله ما مرور میکنیم مسئله از تولید کردن روتینهای سورت سرعت بالا را. یک تفاوت ما بین سورت کردن و پیاده سازی الگوریتم پیاده سازی شده بوسیله بوسیله تولیدات کتابخانه ای فقط اشاره کرده هست این اجرا از الگوریتمها انها انجام ابزار هست به طور کامل تعیین شده بوسیله مشخصاتی از ماشین هدف و اندازه ای از داده ورودی اما نیست بوسیله دیگر مشخصات از داده ورودی. به هر حال در حالتی از سورت اجرا همچنین وابسته است در دیگر فاکتورهای مثل توزیع داده برای سورت شدن. در حقیقت بحث پایین سورت مرج چند راهی را اجرا میکند بسیار خوب در بعضی کلاسهایی ازمجموعه های داده ورودی که radix سورت اجرا میکند بطور غیر کافی در این مجموعه. برای دیگر کلاسهای مجموعه داده ما رعایت میکنیم موقعیت معکوس را. بنابراین دیدگاه تولید کننده های امروزی هست مفید برای بهینه سازی مقادیر پارامتر از الگوریتمهای سورت اما نیست برای انتخاب بهترین الگوریتم برای گرفتن ورودی. برای تبدیل به مشخصات از مجموعه ورودی در [14] ما استفاده کردیم توزیع ای از داده ورودی برای انتخاب الگوریتم سورت. اگر چه این دیدگاه هست ثابت شده که بکلی موثر است اجرای اخر هست محدود شده با اجرایی از الگوریتمهای سورت مثل سورت مرج چند راهی وسورت سریع و سورت radix هستند انتخاب شده در [14] که میتواند انتخاب شده باشد در زمان اجرا.

در این مقاله ما ادامه میدهیم و عمومیت میدهیم به دیدگاه سادهترمان[14] . تولید کتابخانه جدید ما تولید میکند اجرایی از الگوریتمهای سورت مرکب رادر فرمی از یک سلسله مراتبی از سورتهای اولیه که مخصوص شکل نهایی توسعه شده در چهره سلسله مرتبی از ماشین هدف و مشخصات داده ورودی. درک مستقیم باقی کار این است که الگوریتمهای سورت مختلف اجرا میکنند به صورت متفاوت توسعه دادن را در مشخصات ای از هر بخش و مثل یک نتیجه و الگوریتم سورت بهینه می بایستی باشد ترکیبی از این الگوریتمهای سورت مختلف. گذشته از این سورتهای اولیه تولید کردند کد شامل انتخاب اولیه که به صورت داینامیک انتخاب میکند مخلوط الگوریتم مثل یک تابع از مشخصات ای از داده در هر بخش را. در مدت زمان نصب دیدگاه کتابخانه جدید ما جستجو میکند برای تابعی که نگاشت میکند مشخصات ای از

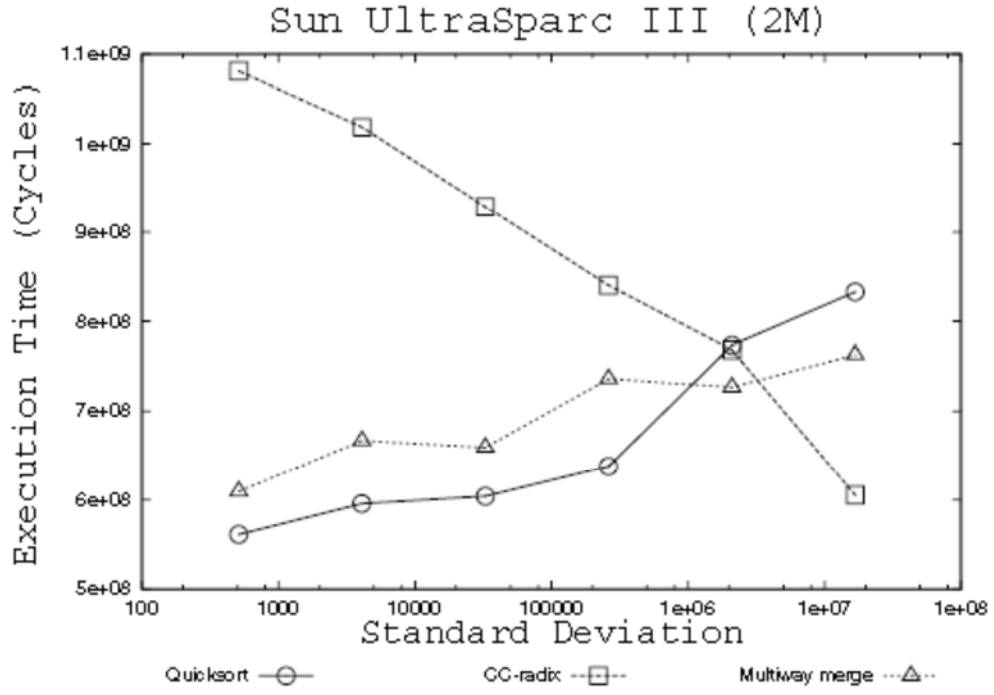
ورودی را برای بهترین الگوریتمهای سورت با استفاده از الگوریتمهای ژنتیک [3,8,16,22]. الگوریتمهای ژنتیک استفاده شده اند برای جستجو برای اختصاص دادن فرمول در [19]SPIRAL و برای بهینه سازی کامپایلر رسمی [4,6,20].

نتایج ما نشان میدهد که دیدگاهمان هست بسیار موثر. بهترین الگوریتم ما داریم تولید شده هست در معدل 36٪ سریعتر از بهترین روتین سورت خالص و شروع با 45٪ سریعتر. روتین سورت ما اجرا میکند یهتر از تمام کتابخانه های تجاری که ما سعی میکنیم شامل IBM ESSL,INTEL MKL و STL از ++C. در معدل روتینهای تولید شده 26٪ و62٪ سریعتر از IBM ESSL در یک IBM PAWR3 و IBM POWER 4 به ترتیب.

نتیجه از این مقاله سازمان دهی شده مثل زیر. بخش 2 بحث میکند اولیه ای که ما استفاده کردیم برای ساختن الگوریتمهای سورت. بخش 3 مرور میکند چرا ما اتنخاب کردیم الگوریتمهای ژنتیک را برای جستجو و مرور بعضی جزییات از الگوریتم ای که ما انجام دادیم. بخش 4 نشان میدهد نتایج اجرا شده را. بخش 5 خلاصه مطالب چگونگی استفاده الگوریتمهای ژنتیک را برای تولید یک سیستم طبقه بندی برای روتینهای سورت میباشد و سرانجام بخش 6 ارائه میکند نتایجمان را.

2 سورت اولیه

در این بخش ما توضیح میدهیم بلوکهای ساخته شده از الگوریتمهای سورت ترکیبییمان را. این اولییه ها انتخاب شده اند بر مبنای ازمایش با الگوریتمهای سورت مختلف و مطالعه ای از فاکتورهای که اثر میکنند به اجرایشان. یک خلاصه از نتایج این ازمایشات در شکل 1 ارائه شده است که کشیده شده زمان اجرا از سه الگوریتم سورت در برابر انحراف معیار از کلیدهای سورت شده.



ترجمه فا

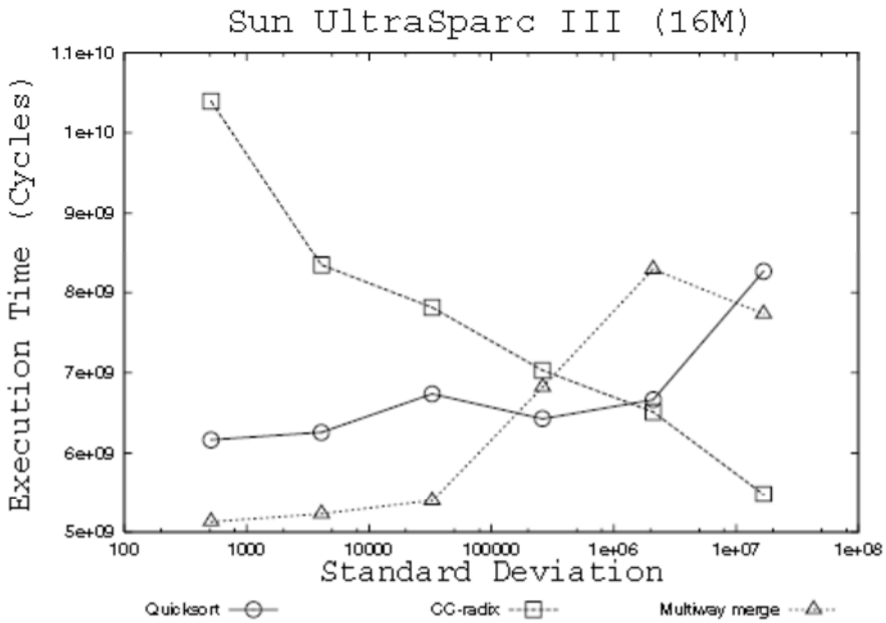


Figure 1: Performance impact of the standard deviation when sorting 2M and 16M keys.

نتایج نشان داده شده برای SUN ULTRASPARC III و برای دو اندازه از مجموعه داده 2 میلیون و 16 میلیون. سه الگوریتم هستند: QUICKSORT [10,17] و CACHE-CONSCIOUS RADIX SORT (CC-RADIX)[11] و MULTIWAY MERG SORT[13]. شکل 1 نشان میدهد که برای 2M رکورد بهترین الگوریتم سورت هست QUICKSORT یا CC-RADIX زمانی که برای 16M رکورد MULTIWAY MERGE SORT یا CC-RADIX هستند بهترین االگوریتم. مشخصات ورودی که تعیین شده زمانی که CC-RADIX هست بهترین الگوریتم هست انحراف معیار از رکوردها برای سورت شدن. CC-RADIX هست بهتر زمانی که انحراف معیار از رکوردها هست بالا برای اینکه اگر مقادیر از عناصر در داده ورودی هست متمرکز شده اند گرداگرد بعضی مقادیر ان هست بسیار شبیه به اینکه بیشتر عناصر در بعضی از BUCKET ها باشند . بنابراین بیشتر پارتیشنها سبقت میگیرند برای بکار بردن قبل از BUCKET های مناسب داخل کش و بنابراین بیشتر کش های فاقد هستند موجب شده در مدت پارتیشن بندی. نتایج اجرا در دیگر پلتفرمها نشان میدهد که تمایل عمومی از الگوریتمها هست همیشه یکسان اما نقطه متقاطع اجرا اتفاق می افتد در نقاط مختلف در پلت فرمهای مختلف.

ان میفهماند برای بسیاری سالها که اجرا QUICKSORT میتواند بهبود شود زمانی که ترکیب شده با دیگر الگوریتمها [17]. ما تایید میکنیم به صورت ازمایشی که وقتی پارتیشن هست کوچکتر از یک استانه خوب (که مقدار وابسته است در پلتفرم هدف) ان هست بهترین برای استفاده INSERTION SORT یا ذخیره داده در رجیسترها و سورت بوسیله تعویض مقادیر ما بین ریجسترها [14] به جای ادامه دادن به صورت بازگشتی بکار بردن quicksort. Register sort هست یک الگوریتم کد مستقیم که اجرا میکند مقایسه-و-تعویض از مقادیر سورت شده در رجیسترهای پروسسور [13].

ترجمه فا

[5] Darlington معرفی کرده ایده ای از سورتهای اولیه و تشخیص merge sort و quicksort مثل دو الگوریتم اولیه. در این مقاله ما جستجو کیکنیم برای یک الگوریتم بهینه بوسیله ساختن الگوریتمهای سورت ترکیبی. ما استفاده کردیم دو نوع از اولیه برای ساختن الگوریتمهای سورت تازه: سورت کردن و انتخاب اولیه. سورتهای اولیه ارائه میکنند یک الگوریتم سورت خالص را که شامال پارتیشن بندی داده است به طوریکه مثل radix sort و merge sort و quicksort. انتخاب اولیه ارائه میکند یک پروسه برای اجرا شدنه در زمان اجرای که به صورت داینامیک تصمیم میگیرد که الگوریتم سورت را برای بکار بردن.

الگوریتمهای سورت ترکیبی مطرح میکنند در این مقاله فرض شده است که داده هست سورت شده در پی در پی حافظه محلی. داده هست به صورت بازگشتی پارتیشن شده بوسیله یکی از 4 متدهای پارتیشن بندی. پارتیشن بندی بازگشتی خاتمه میدهد زمانی که یک الگوریتم سورت شکل گرفت هست بکار میرود در پارتیشن. ما الان توضیح میدهیم 4 پارتیشن بندی اولیه را در زیر بوسیله یک توضیح از دو سورت اولیه شکل گرفته. برای هر اولیه ما همچنین تشخیص میدهیم مقادیر پارامتری که باید باشد جستجو شده با تولیدات کتابخانه.

1- تقسیم-با-مقدار (DV)

این اولیه مرتبط است با فاز اول از QUICKSORT که در حالتی از یک پارتیشن دودویی انتخاب میکند یک PIVOT و دوباره سازمان میدهد داده را مثل قسمت اول از بردار شامل کلیدها با مقادیر کوچکتر از PIVOT و قسمت دوم انهایی که بزرگتر یا مساوی PIVOT هستند. در کار ما اولیه DV میتواند پارتیشن کند مجموعه ای از رکوردها را داخل دو یا چند قسمت با استفاده از یک پارامتر NP که مشخص میکند تعداد PIVOT ها را. بنابراین این اولیه توضیح میدهد تقسیم کردن مجموعه ورددی را داخل NP+1 پارتیشن و دوباره میچیند داده را گرداگرد PIVOT NP ها.

2-تقسیم- با – موقعیت (DP)

این اولیه مرتبط میکند با MULTIWAY MERGE SORT و گام اولیه میشکند ارایه ورودی را از کلیدها داخل دو یا چند پارتیشن یا زیر مجموعه ای از اندازه یکسان. ان هست صریحا در DP اولیه که بهد از تمام پارتیشنها دارد پردازش میکند پارتیشنها مرج شده اند با بدست اوردن یک ارایه سورت شده . مرج کردن هست انجام گرفته با یک HEAP یا صف اولویت[13]. عمل مرج مثل زیر کار میکند. در شروع کردن برگهای از HEAP هستند عناصر اولیه از هر پارتیشن. سپس جفت برگها هستند مقایسه مشده اند کوچکترین ترفیع داده میشود به نود پدر و یک عنصر جدید از پارتیشن که شامل شده عنصر ترفیع داده شدهمیشود یک برگ. این هست انجام به صورت بازگشتی تا زمانی که HEAP پر شود. بهد از این عنصر در بالایی از HEAP هست استخراج شده جایگزین شده در بردار مقصد یک عنصر تازه از مرتبط ساختن زیر مجموعه هست ترفیع داده شده و ژروسه تکرار شده دوباره شکل 2 نشان میدهد یک تصویر از HEAP را.

HEAP ساخته شده مثل یک ارایه که برادر ها هستند جا گرفته در موقعیت پی در پی. زمانی که مرج کردن استفاده میشود HEAPرا عمل پیدا کردن فرزند با کوچکترین کلید هست اجرا شده به صورت تکراری. اگر تعداد از فرزندان از هر پدر هست کوچکتر از تعدادی از نودها که مناسب است در یک خط کش , خط کش خواهد شد زیر استفاده شده.

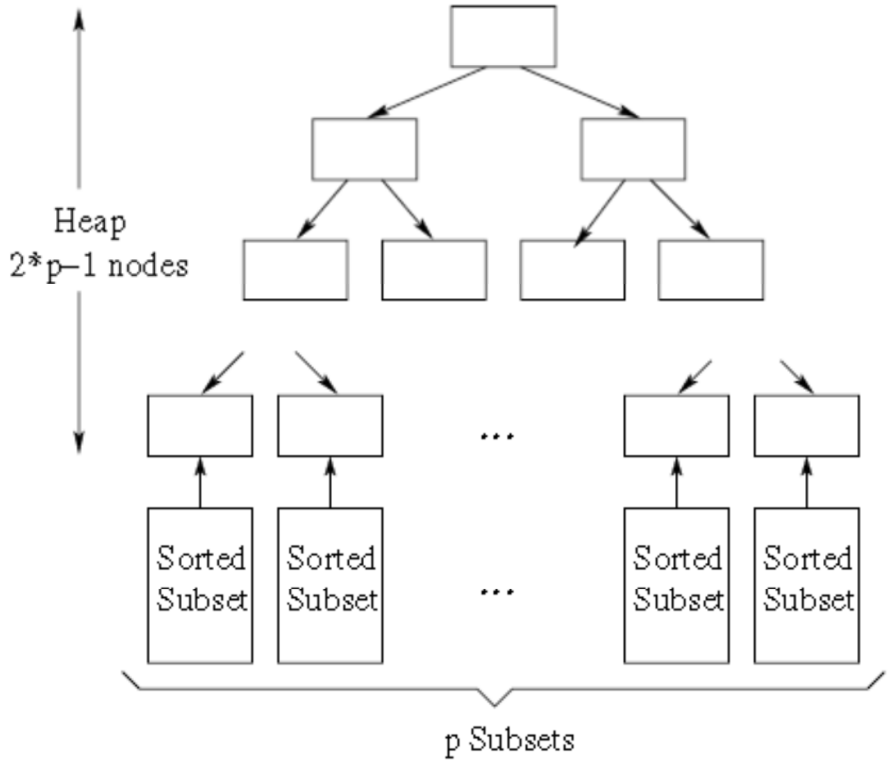


Figure 2: Multiway Merge.

برای حل کردن این مسئله ما استفاده کردیم HEAP با یک گنجایش خروجی که هست یک چندگانه از A/r که A هست اندازه ای از خط کش و r هست اندازه ای از هر نود. این هست هر پدر از HEAP ما دارد A/r فرزند [14]. این میگیرد ماکزیمم معدل از محل فضایی. البته برای این هست درست ساختار ارایه ساخته میشود HEAP نیاز دارد برای ردیف کردن شایسته.

DP اولیه دارد دو پارامتر: اندازه که مشخص میکند اندازه ای از هر پارتیشن و FANOUT گنجایش خروجی این مشخص میکند تعداد فرزند از هر نود از HEAP.

3-تقسیم با – RADIX (DR)

تقسیم با RADIX اولیه مرتبط میکند با یک گام از RADIX SORT الگوریتم. DR اولیه توزیع میکند رکوردها را برای سورت شدن داخل BUCKET ها وابسته میشود در مقدار ای از یک عدد در رکورد. بنابراین اگر ما استفاده میکنیم یک RADIX از r بیت رکوردها توزیع خواهند شد داخل 2^r تا sub-bucket ها مبنا شده در مقداری از یک عدد از r بیت. پیاده سازی ما تکیه دارد در الگوریتم ترازی [13] که برای هر عدد پردازش میکند در سه گام: گام اول مقایسه میکند یک هیستوگرام با تعداد ای از کلیدها توسط BUCKET دومی مقایسه میکند مجموع جزیی را که مشخص میکند محلی در بردار مقصد را که هر BUCKET شروع کرده و یک گام اخر حرکت میدهد کلیدها را از بردار منبع به مقصد.

DR اولیه دارد یک پارامتر RADIX که مشخص میکند اندازه ای از RADIX در تعدادی از بیتها را. موقعیت عدد در رکورد نیست مشخص شده در اولیه اما هست تعیین شده در زمان اجرا مثل زیر. به صورت ادراکی یک شمارنده هست نگه داشته شده برای هر پارتیشن. شمارنده مشخص میکند موقعیتی که عدد استفاده شده است برای RADIX SORT شروع شده. هر پارتیشن که هست ساخته شد به ارث میبرد تعدادی از پدرانش را. شمارنده هست مقدار دهی اولیه شده در صفر و هست افزایش داده شده بوسیله اندازه از RADIX (در تعدادبیتها) هر زمان یک DR اولیه بکار برده شده باشد.

4- تقسیم –با – RADIX – با فرض – یکنواخت – توزیع شده(DU)

این اولیه هست مبنا شده در اولیه DR پیشین اما فرض میکند که یک عدد هست بطور یکسان توزیع شده است. مقایسه از هیستوگرام وگامهای مجموع جزیی در DR اولیه استفاده شده است برای تعیین کردن تعدادی از کلیدها از هر مقدار ممکن و رزرو میکند مرتبط ساختن فضا را در بردار خروجی. به هر حال این گامها (بویژه مقایسه کردن هیستگرام) هست بسیار گران. برای اجتناب این سربار ما میتوانیم فرض کنیم که یک عدد هست بهطور یکسان توزیع شده و این تعداد از کلیدها برای هر مقدار ممکن هست یکسان. بنابراین با DU اولیه زمانی که سورت کردن یک ورودی با n کلید و یک RADIX از اندازه r هر SUBBUCKET هست فرض شده با شامل شدن n/2^r کلید. در تمرین بعضی subbucket سرریز خواهد کرد فضای رزرو شده برای اینکه توزیع از بردار ورودی نیست کلا یکسان. به هر حال اگر سربار برای هندل کردن حالات زمانی که اینجا هست سربار کمتر از بالایی برای مقایسه هیستوگرام و گام انباشتگی DU اولیه اجرا خواهد شد سریعتر از یک DR. مثل در DR , DU اولیه دارد یک پارامتر RADIX.

یک قسمت از این اولیه ها ما داریم اولیه بازگشتی بکار برده خواهند شد تا زمانی که پارتیشن ذخیره شده باشد. ما مینامیم انها را اولیه های برگ.

5-برگ-تقسیم-با-مقدار (LDV)

این اولیه مشخص میکند که DV اولیه باید بکار برده شود به صورت بازگشتی با سورت پارتیشنها. به هر حال زمانی اندازه ای از پارتیشن هست کوچکتر از یک استانه درست این LDV اولیه استفاده میکند یک الگوریتم

سورت رجیستر در جا را برای سورت رکوردها در پارتیشن. LDV دارد دو پارامتر: np که مشخص میکند تعدادی از PIVOT ها مثل DV اولیه و THRESHOLD که مشخص میکند اندازه پارتیشن در زیر که الگوریتم سورت ریجیستر هست بکار برده شده.

6- برگ-تقسیم-با-RADIX (LDR)

این اولیه مشخص میکند که DR اولیه استفاده کرده است با سورت زیر مجموعه های باقیمانده. LDR دارد دو پارامتر: RADIX و THRESHOLD. مثل در LDV فاقشاخمی استفاده کرده است برای مشخص کردن اندازهی از پارتیشن که الگوریتم سویچ میکند به سورت رجیستر.

یادداشت که اگرچه تعداد نوع از سورت اولیه میبایستی متفاوت باشد ما داریم انتخاب برای استفاده این 6 تا برای اینکه انها ارائه میکنند الگوریتمهای خالص که فراهم کرده است بهترین نتایج را در ازمایشمان. الگوریتمهای سورت دیگر مثل SHELL SORT هرگز فراهم نمیکنند اجرایی از الگوریتمهای سورت انتخاب شده اینجا. به هر حال انها بایستی شامل شوند در استخوان بندییمان.

تمام سورتهای اولیه پارامترهایی دارند که بیشتر مقدار اختصاصی وابسته خواهند شد در چهره سلسله مراتبی از ماشین هدف. ملاحظه کنید برای مثال DP اولییه. پارامتر SIZE هست مرتبط با اندازهای از کش زمانی که گنجایش خروجی مرتبط شده در تعدادی از عناصر که مناسب در یک خط کش. به طور مشابه np و RADIX از DV و DR اولیه هستند مرتبط با اندازه کش. به هر حال مقدار دقیق از این پارامترها نمیتواند به اسانی تعیین کند یک تقدم. برای مثال رابطه ما بین np و اندازه کش نیست سرراست و مقدار بهینه شاید تغییر بدهد وابسته بودن را در تعدادی از کلیدهای سورت. پارامتر THRESHOLD هست مرتبط با تعدادی از رجیسترها.

در ادامه برای اولیه های سورت ما استفاده میکنیم اولیه های انتخاب را. اولیه های انتخاب هستند استفاده شده در زمان اجرا برای تعیین کردن, مبنا شده در مشخصاتی از ورودی اولیه سورت برای بکار بردن برای هر زیرپارتیشن از یک پارتیشن گرفته شده. مبنا در نتایج نشان داده شده در شکل 1 اینجا اولیه های انتخاب طراحی شده اند برای گرفتن داخل اکانت تعدادی از رکوردها در پارتیشن AND/OR انحراف معیارشان. این انتخابهای اولیه هستند:

1-انشعاب – با – سایز (BS)

مثل انچه نشان داده شده در شکل 1 تعداد رکوردها برای سورت هست یک مشخصات ورودی که تعیین میکند رابطه اجرا از سورت اولیه مان را. این اولیه BS استفاده شده است برای انتخاب مسیر مبنا از اندازه پارتیشن. بنابراین این اولیه BS دارد یک یا بیشتر (SIZE1,SIZE2,...) پارامترها برای انتخاب مسیر برای پیروی. مقادیر اندازه سورت شده اند و استفاده شده اند برای انتخاب n+1 ممکنات (کمتر از SIZE1 مابین SIZE1 و SIZE2و ... بزرگتر از SIZEn).

2- انشعاب – با – انتروپی (BE)

گذشته از این اندازه ی پارتیشن دیگر مشخصات ورودی که تعیین میکنند اجرا را روی سورت اولیه هست انحراف معیار. به هر حال به جای استفاده کردن انحراف معیار برای انتخاب مسیرهای مختلف برای پیروی کردن ما استفاده میکنیم مثل انچه در [14] انجام شده و ادراک از انتروپی از تئوری اطلاعات.

اینجا چندین علت برای استفاده انتروپی به جای انحراف معیار وجود دارد. انحراف معیار هست گران برای محاسبه به طوریکه ان نیاز دارد به چندین عملگرهای نقطه شناور پیش رکورد. اگر چه مثلا میتوان دید در شکل 1 انحراف معیار هست فاکتوری که تعیین کردن ان در زمان CC-RADIX هست بهترین الگوریتم و در تمرین رفتاری از CC-RADIX وابسته است بیشتر از انحراف معیار از رکوردها برای سورت و هر چقدر بیشتر مقادیر از هر عدد هستند گسترده بیرون. انتروپی از یک موقعیت عدد خواهد داد به ما این اطلاعات را. سورت CC-RADIX توزیع کرده است رکوردها را برای تطبیق برای مقدار از یکی از عددها. اگر مقادیر از این عدد هستند باز پخش شده انتروپی خواهد بود بالا و اندازه ها از نتیجه گیری SUBBUCKET ها خواهد بود بسته برای یکدیکر و همچنین یک نتیجه تمام SUBBUCKETها خواهند بود بیشتر شبیه برای مناسب بودن در کش. به صورت نتیجه بخش هر SUBBUCKET میبایستی باشد به طور کامل سورت شده با از دست دادنهای کش کمی. اگر به هر حال انتروپی هست پایین بیشتر از رکوردها پایان خواهند یافت در SUBBUCKET یکسان که افزایش احتمال که یک یا بیشتر SUBBUCKET ها نیستند مناسب در کش. برای سورت کردن این SUBBUCKETها میبایستی لازم باشد بسیاری فاقد کش.

برای محاسبه انتروپی در زمان اجرا ما نیاز داریم به اسکن مجموعه ورودی و محاسبه تعداد کلیدها که دارند یک مقدار ویژه برای هر موقعیت عدد. برای هر عدد انتروپی هست کامل شده مثل $\sum_i -P_i * \log_2 P_i$, که $P_i = c_i/N$ و c_i هست تعدادی از کلیدها با مقدار i در این عدد و N هست مجموع تعداد کلید . نتیجه هست یک بردار از انتروپی ها که هر عنصر از بردار ارائه میکند انتروپی از یک موقعیت عدد در کلید. ما سپس محاسبه میکنیم یک مقدار مقیاس انتروپی S, مثل تولید داخلی از بردار انتروپی محاسبه شده (Ei) و یک بردار وزن Wi: $S = \sum_i E_i * W_i$. نتایج مقدار S هست استفاده شده برای انتخاب مسیر برای پردازش کردن با سورت. مقدار عددی انتروپی و بردار وزن هستند مقادیر پارامتر لازم شده برای این اولیه. بردار وزن نشان میدهد شدتی از هر عدد در اجرایی از سورت RADIX. در مدت فاز تلاش ان میتواند اپدیت شده باشد با اجرای داده با استفاده الگوریتم غربال (WINNOW). جزییات بیشتر میتواند پیدا شود در [14].

Type	Prim.	Parameters
Sorting	*DV*	*np*, number of pivots
	DP	*size*, partition size *fanout* of the heap
	DR	*radix* size in bits
	DU	*radix* size in bits
	LDV	*np*, number of pivots *threshold* for in-place register sort
	LDR	*radix* size in bits *threshold* for in-place register sort
Selection	*BS*	n, there are $n+1$ branches *size*, n size-thresholds for the $n+1$ branches
	BE	n, there are $n+1$ branches *entropy*, n scalar-entropy-value-thresholds for the $n+1$ branches and the weight vector.

Table 1: Summary of primitives and their parameters.

اولیه ها و پارامترهایشان لیست شده اند در جدول 1. ما استفاده خواهیم کرد 8 اولیه ارائه شده در اینجا را (6 سورت اولیه و دو انتخاب اولیه) برای ساختن الگوریتمهای سورت. شکل 3 نشان میدهد یک مثال که الگوریتمهای سورت مختلف رمزی کردند مثل درختی از اولیه ها. شکل 3 نشان میدهد انکد شدن متناظر برای یک الگوریتم سورت RADIX خالص که تمام پارتیشنها سورت شده اند با استفاده یکسان RADIX از 2^5. سورتهای اولیه DR ما داده را تطبیق میدهند با مقداری از چپترین عددی که ندارد هنوز پردازش نشده است. شکل 3-(b) نشان میدهد انکد کردنی از یک الگوریتم که پارتیشنهای اولیه را تطبیق میدهد با مقداری از چپترین مبنا 2^8 عدد و سپس سورت میکند هر BUCKET نتیجه شده را با استفاده از سورت RADIX با اندازه RADIX از چه 2^8 یا 2^4 مرتبط باشد در تعداد ای از رکوردها از هر BUCKETهای تولید شده بوسیله top-level-radix-sort. Radix 2^4 هست استفاده شده زمانی که تعداد رکوردها هست کمتر از S1 و 2^8 در حالت دیگر. یادداشت که زمانی که پارتیشنهای نتیجه شده دارند اندکی از 16 عنصر الگوریتم in-place-register-sorting بکار برده شده است. شکل 3-(c) نشان میدهد انکود کردن از یک الگوریتم پیچیده را. مجموعه ورودی هست به صورت اولیه پارتیشن بندی شده داخل زیر مجموعه های با 32k عنصری . برای هر پارتیشن انتروپی هست محاسبه شده مثل توضیح داده شده بالا و مبنا شده در مقدار محاسبه شده یک

الگوریتم متفاوت هست بکار برده شده است. اگر انتروپی کمتر از V1 باشد یک quicksort بکار برده شده است. این quicksort تمایل دارد به یک in-place-register-sort کردن زمانی که پارتیشن شامل 8 یا کمتر عنصر باشد. اگر انتروپی هست بیشتر از V2 (with:V2>V1) یک سورت RADIX استفاده میکند RADIX 2^8 بکار برده شده است. در حالت دیگر اگر انتروپی هست مابین v1 و v2 انتخاب دیگری هست ساخته شده بر مبنای اندازه پارتیشن. اگر اندازه کمتر از S1 باشد یک سورت radix با radix 2^8 بکار برده میشود. در حالت دیگر یک three-way-quicksort بکار برده میشود. در پایان هر زیر مجموعه سورت شده است اما انها نیاز دارند برای سورت شدن در میان خودشان. برای این منظور زیر مجموعه های ابتدایی هستند مرج شده با استفاده از یک heap مثل انی که در شکل 2 است و با یک fanout ای از 4 که هست مقدار پارامتر از DP اولیه.

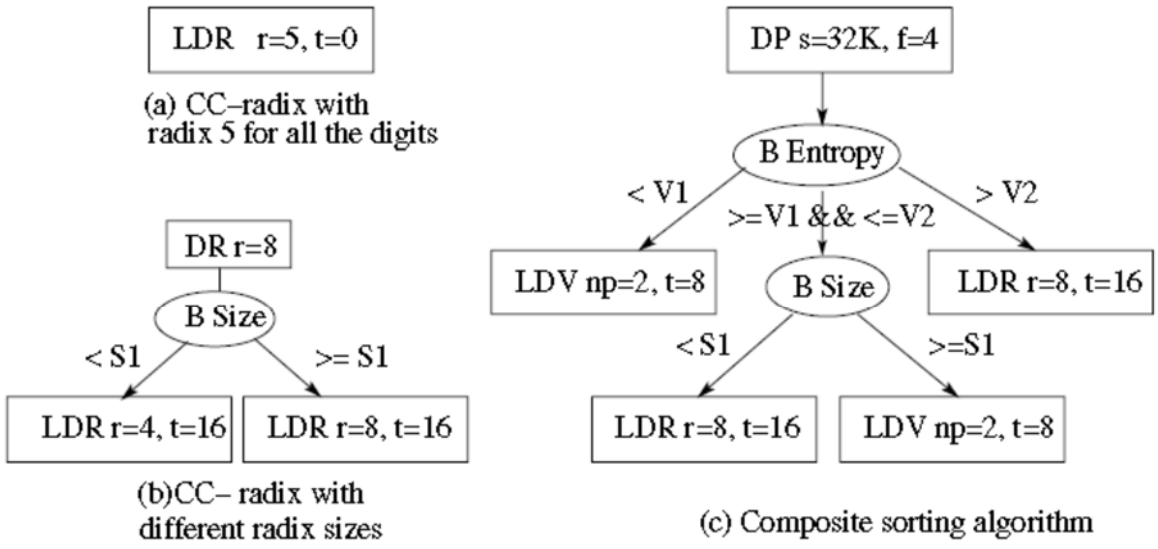


Figure 3: Tree based encoding of different sorting algorithms.

اولیه هایی که ما استفاده کردیم نمیتوانند تولید کنند تمام الگوریتمهای سورت ممکن را اما با ترکیب کردن شان انها میتوانند بسازند یک فضای بسیار بزرگتر از الگوریتمهای سورت که این شامل میشود تنها الگوریتمهای سورت خالص متداول را شبیه quicksort یا radix سورت همچنین با تغییر پارامترها در سورت کردن و اولیه های انتخاب ما میتوانیم تبدیل کنیم به معماری از ماشین هدف و برای مشخصاتی از داده ورودی.

genesort-1

ترجمه فا

در این بخش ما مرور میکنیم استفاده از الگوریتمهای ژنتیک را برای بهینه سازی سورت. ما در ابتادا مرور میکنیم چرا ما باور داریم که الگوریتمهای ژنتیک هستند یک استراتژی جستجوی خوب و سپس ما مرور میکنیم چطور از انها استفاده کنیم.

3-1 چرا الگوریتمهای ژنتیک را استفاده میکنیم؟

متداولا پیچیدگی از الگوریتمهای سورت که مطالعه شده بود در مدتی از تعداد مقایسات اجرا شده اند با فرض کردن یک توزیع مشخص از ورودی مثل توزیع یکنواخت [13]. مطالعات فرض میکند که زمان دسترسی هر عنصر یکسان میباشد. این فرض به هر حال نیست درست در پروسسورهای امروزی که دارند یک کش عمیق سلسله مراتبی و ترکیب معماری پیچیده. به طوریکه انها نیستند مدلهای انالیزی از اجرای الگوریتمهای سورت در مدت ترکیب معماری از ماشین, تنها راه برای شناسایی بهترین الگوریتم هست با جستجو کردن.

دیدگاه ما هست برای استفاده از الگوریتمهای ژنتیک برای جستجو کردن برای برای یک الگوریتم سورت بهینه. فضای جستجو هست تعریف شده با ترکیبی از سورت کردن و انتخاب کردن اولیه توضیح داده شده در بخش 2 و مقادیر پارامترها از اولیه ها. هدف از جستجو هست برای شناسایی سورت سلسله مراتبی که بهترین ترکیب معماری مناسب از ماشین و مشخصات ای از مجموعه ورودی.

اینجا چندین دلیل هست که چرا ما داریم انتخاب میکنیم الگوریتمهای ژنتیک را برای اجرای جستجو.

- برای استفاده کردن اولیه ها در بخش 2 الگوریتمهای سورت میتوانند انکود شده باشند مثل یک درخت در شکل 3. الگورتمهای سورت میتوانند باشند به اسانی استفاده شده برای جستجو در این فضا برای بیشتر اختصاص دادنه فضای درخت و مقادیر پارامترها.
- فضای جستجو از الگوریتمهای سورت که میتواند مشتق شده باشد با استفاده از 8 اولیه در بخش 2 هست همچنین بزرگتر برای جستجوی هییوریستیک.
- الگوریتمهای ژنتیک حفظ میکنند زیر درختهای بهتر را و میگیرند زیر درخت شانس بیشتر را برای تولید دوباره. الگوریتمهای سورت میتواند بگیرد مزایای از اینکه یک زیر درخت هست همچنین یک الگوریتم سورت.

در حالت ما برنامه نویسی ژنتیک نگه داری میکند یک جمعیت از ژنهای درخت را. هر ژن درخت هست یک بیان که ارائه میکند یک الگوریتم سورت. احتمالی که یک ژن درخت انتخاب شده باشد برای تکثیر (نامیده شده CROSSOVER) هست مناسب با سطحی از FITNESS (شایستگی). ژنهای بهتر میگیرند فرصت بیشتر برای تولید فرزندان. برنامه نویسی ژنتیک همچنین هست تصادفی تغییر میدهد بعضی عبارات را برای ساختن یک ژن بهتر ممکن شدنی را.

3-2 بهینه سازی از سورت با الگوریتمهای سورت

3-2-1 انکد کردن

مثل بحث شده بال ما استفاده میکنیم یک شما بر مبنای درخت که نودها از درخت هستند سورت کردن و انتخاب اولیه.

3-2-2 عملگرها

عملگرهای ژنتیک استفاده شده اند برای مشتق کردن اولاد تازه و معرفی شانس در جمعیت. CROSSOVER و MUTATION هستند دو عمگری که الگوریتمهای ژنتیک بیشتر استفاده میکنند. CROSSOVER تغییر میدهد زیر درختها را از درختهای متفاوت. عملگر MUTATION بکار میبرد عوض کردن را برای یک درخت تنها. بهدا ما مرور میکنیم چطوری بکار ببریم این دو عملگر را.

CROSSOVER

منظور از CROSSOVER هست برای تولید اولادی که دارند بهترین اجرا را از پدرشان. این هست شبیه به اینکه برای اتفاق افتادن زمانی که اولاد به ارث میبرند زیر درختهای بهتر را از پدران. در این مقاله ما استفاده میکنیم SIGNAL-POINT-CROSSOVER را و ما انتخاب میکنیم CROSSOVER POINT تصادفی. شکل 4 نشان میدهد یک مثال از SINGL-POINT-CROSSOVER.

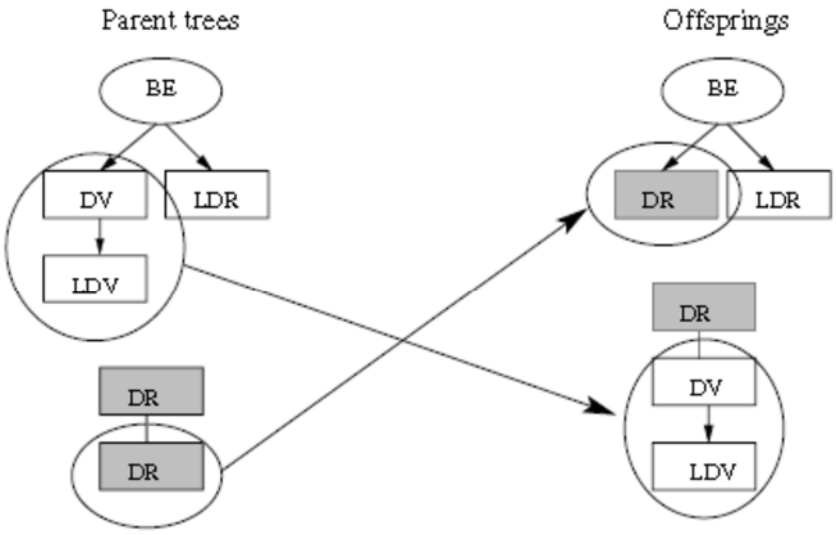


Figure 4: Crossover of sorting trees.

MUTATION

MUTATİON کار میکند در یک درخت تنها که ان تولید میکند بعضی عوض کردنها و مطرح میکند تنوع را در جمعیت. MUTATION جلوگیری میکند از جمعیت از نگهداری یکسان بعد از هر تولید ویژه [1] .این دیدگاه برای بعضی مقدار اجازه میدهد جستجو را برای فرار از بهینه محلی. MUTATION تعویض میکند مقادیر پارامترها را به امید اینکه پیدا کند بهترینها را. عملگر MUTATION ما میتواند اجرا کند تعویضهای زیر را:

1.تعویض کردن مقادیر از پارامترها در نودهای سورت کردن و انتخای اولیه. پارامترها تعویض میکنند به صورت تصادفی اما مقادیر تازه هستند محدود به قدیمیها.

2.تعویض دو زیر درخت. این نوع از MUTATION میتواند کمک کند در حالات شبیه به انی که نشان داده شده در شکل 5-(a) که یک زیر درختی که هست خوب برای مجموعه سورت کمتر از 4M رکورد هست بکار برده شده برای مجموعه های بزرگتر. با تعویض کردن زیر درخت ما میتوانیم اصلاح کنیم این نوع از عوضی جایگزین شده را.

3. افزودن یک زیر درخت تازه. این نوع از MUTATION هست مفید زمانی که بیشترین پارتیشن بندی هست مورد نیاز در امتداد یک مسیر از درخت. شکل 5-(b) نشان میدهد یک مثال از این MUTATION را.

4. حذف یک زیر درخت. زیر درختهای غیر لازم میتوانند باشند حذف شده با این عملگر.

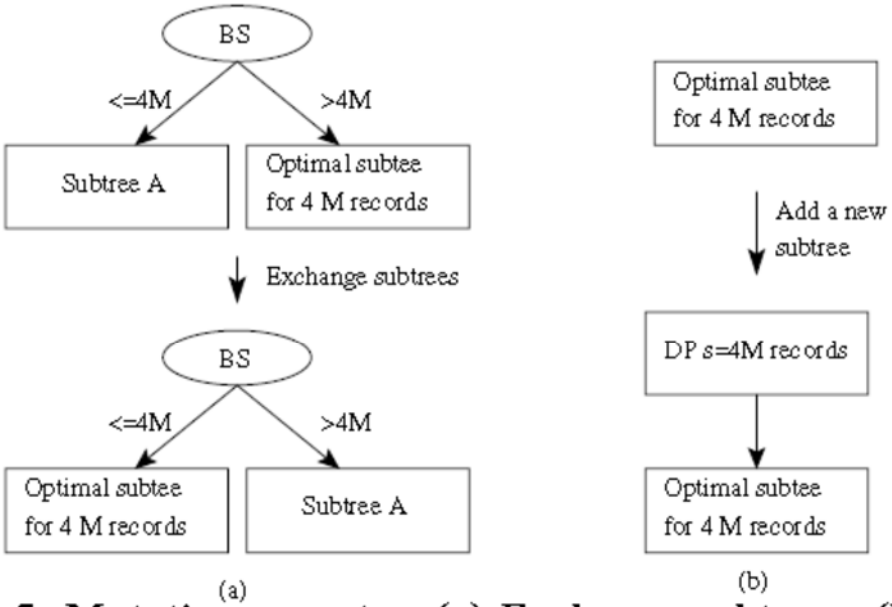


Figure 5: Mutation operator. (a)-Exchange subtrees. (b)-Add a new subtree.

3-2-3 تابع سایستگی FITNESS

تابع FITNESS تعیین میکند احتمالی از فرد برای تکثیر را. بیشترین FITNESS از یک فرد بیشترین تعویض ان تکثیر خواهد شد و تغییر خواهد کرد.

در حالت ما اجرا استفاده شده خواهد شد مثل تابع FITNESS. به هر حال دو ملاحظه زیر گرفته شده است داخل اکانت در طراحی از تابع FITNESS ما:

1.ما جستجو میکنیم برای یک الگوریتم سورت که اجرا میکند به خوبی سراسر تمام ورودیهای ممکن. بنابراین مینگین اجرا از یک ردخت هست FITNESS پایه اش. به هرحال همان طور که ما خواستیم الگوریتم سورت

برای اجرای سازگار با استفاده چند گانگی FITNESS پایه با یک فاکتور که وابسته است به انحراف معیار از اجرایش زمانی که سورت کردن ورودیهای تستی.

2.در اولین تولید واریانس FITNESS از جمعیت هست بالا این است یک کمی درختهای سورت کننده دارد یک اجرای بسیار بهتر از دیگران. اگر تابع FITNESS ما بود مستقیما متناسب با اجرایی از درخت, بیشتر اولاد میبایستی زاده شده باشند از این درختهای کم, به طوریکه انها میبایستی باشند یک تناسب بسیار بالا برای تکثیر. در نتیجه این اولاد میبایستی به زودی تصرف کنند بیشتر جمعیت را. این نتیجه میتواند باشد درهمگرایی قبل از موقع, که میبایستی جلوگیری کند سیستم از کاوش ارایه های فضای جستجو بیرون از همسایگی درخت شایسته بسیار بالا. برای ادرس دادن این مسئله تابع FITNESS ما استفاده میکند اجرا را به ترتیب یا در ترتیبی از درختهای سورت در جمعیت. با استفاده کردن اجرای ترتیبی (رتبه بندی), تفاوت اجرای مستقل ما بین درختها

Genetic Algorithm {

P = Initial Population

While (stopping criteria is false) do {

- Apply mutation and crossover and generate set M of k individuals
- $P = P \cup M$
- S = Input sets with different sizes and different standard deviations
- Use each genome of P to sort each element of S
- Apply fitness function to remove the k least fit individuals from P.

}}

Figure 6: Genetic Algorithm

رسیدگی نشده اند و درختها با اجرای پایین دارند احتمال بالا برای تکثیر سپس اگرمقدار اجرای مستقل استفاده شده باشد. این پرهیز اجرا از همگرایی اولیه و از همگرایی به یک بهینه محلی.

2-3-4 الگوریتم تکاملی (EVOLUTION ALGORITHM)

یک تصمیم مهم هست برای انتخاب الگوریتم تکاملی مناسب. الگوریتم تکاملی تعیین میکند چطور اولاد زیاد تولید خواهند شد چطور فردها بسیار از تولید جاری جایگزین خواهند شد همچنین.

در این کار ما استفاده میکنیم الگوریتم تکاملی STEADY-STATE. برای هر تولید تنها یک تعداد کم از فردهای مناسب کمترین در تولید جاری جایگزین شده اند با اولاد تولید شده تازه. در نتیجه بسیاری از فردها از جمعیت قبلی هستند به طور مشابه باقی میمانند.

شکل 6 نشان میدهد کد را برای الگوریتم تکاملی STEADY-STATE که ما استفاده کردیم برای تولید یک روتین سورت. هر تولید یک تعداد ثابت از اولاد تازه را تولید خواهد کرد اگر چه CROSSOVER و بعضی فردها جهش(تغییر) خواهند کرد مثل توضیح داده شده بالا. تابع FITNESS استفاده خواهد کرد برای انتخاب فردها که عملگر CROSSOVER و MUTATION بکار برده اند. سپس چندین مجموعه ورودی با مشخصات متفاوت (انحراف معیار و تعدادی از رکوردها) تولید خواهند کرد و استفاده میکند برای تربیت کردن درختهای سورت از هر تولید. ورودیهای تازه تولید شده اند برای هر تکرار. اجرا تامین شده است با هر الگوریتم سورتی که استفاده خواهد کرد تابع FITNESS را برای تصمیم گرفتن اینکه کدام هستند فردها با کمترین FIT و پاک میکند انها را از جمعیت. تعداد فردها پاک شده هست یکسان مثل تعداد تولید شده. این راه تعداد فردها باقی میماند ثابت سراسر تولیدات.

چندین ضابطه میتواند انتخاب شده باشد مثل ضابطه نگه داشتن مثل استوپ بعد از یک تعداد از تولید یا استوپ زمانی که اجرا ندارد بهبودی بیش از یک درصد خوب در تعداد تولیدات گذشته. ضوابط نگه داشتن و جمعیت اولیه که ما استفاده خواهیم کرد بحث شده اند در بخش بعدی.

4 – ارزیابی از GENESORT

در این بخش ما ارزیابی میکنیم دیدگاهمان را از استفاده نمدن الگوریمهای ژنتیک برای بهینه سازی الگوریتمهای سورت. در بخش 4-1 ما بحث میکنیم ستاپ محیطی را. بخش 4-2 اراده میکنیم نتایج اجرا را و بخش 4-3 ارائه میکنیم در خت سورت تولید شده برای هر پلتفرم وانالیز مشخصاتش.

4-1 ستاپ محیطی

ما ارزیابی میکنیم دیدگاهمان را در 7 پلتفرم مختلف: AMD ATHLON MP,SUN ULTRASPARC III,SGI R 12000,IBM POWER3, IBM POWER4, INTEL ITANIUM 2, INTEL XEON . جدول 3 لیست شده برای هر پلتفرم پارامترهای مهماری اصلی. سیستم عامل کامپایلر و عملیات کامپایلر استفاده شده اند برای ازمایشات.

برای ارزیابی ما پیروی میکنیم الگوریتمهای ژنتیک را در شکل 6. جدول 2 خلاصه کرده مقادیر پارامتری که ما داریم استفاده میکنیم برای ازمایشات. ما استفاده کردیم یک جمعیت از درختهای سورت 50 تا و ما اجازه دادیم

به انها نمو کنند با استفاده الگوریتم STEADY-STATE برای 100 تولید. به هر حال ازمایشات ما (نشان نمیدهد اینجا برای اینکه از فضای لازم) نشان میدهد که 30 تولید کافی است در بیشترین حالت برای تامین یک حل استوار.

Parameters for Genetic algorithm	
Population Size	50
#Generations	100
#Generated offsprings	30
Mutation Rate Probability	6%
#Training input sets	12

Table 2: Parameters for one generation of the Genetic Algorithm.

الگوریتم ژنتیک ما برای هر دوی ساختار درخت و مقادیر پارامتر جستجو میکند. بنابراین یک نرخ جایگزینی بالا و یک نرخ جهش (MUTATION) لازم هستند برای تعهد که یک مقدار پارامتر اختصاصی میتواند رسیده باشد اگر جه تکامل تصادفی باشد. ما انتخاب کردیم یک نرخ جایگزینی از 60٪ که برای ازمایشات ما, در حدود 30 فرد تازه تولید شده اند با وجود CROSSOVER در هر تولید. عملگر MUTATION تغییر میدهد اولاد تازه را با یک احتمال 6٪. همچنین 12 مجموعه داده مختلف تولید شده اند در هر تولید و استفاده شده اند برای تست 80 در خت سورت (50 پدر + 30 اولاد).

برای جمعیت اولیه ما انتخاب میکنیم درختهایی که ارائه میکنند الگوریتمهای سورت اولیه را زا CC-RADIX[11],QUICKSORT[17],MULTIWAY MERGE[13],ADAPTIVE ALGORITHM که ما ارائه کردیم در [14] و دگرگونیهای انها. در تمام پلتفرمها که ما سعی کردیم فردهای اولیه سریعا جایگزین شده بودند با اولاد بهتر اگر چه بسیاری زیر درختها از جمعیت اولیه هنوز ارائه شده بودن در تولیدات اخر.

زمان برای تولید کردن روتینهای سورت تغییر میدهد پلتفرم را به پلتفرم و رنج میدهد از 9 ساعت در INTEL XEON به 80 ساعت در SGI R 12000.

4-2 نتایج ازماشات

در این بخش ما ارئه میکنیم اجرایی از روتینهای سورت تولید شده با استفاده از الگوریتمهای ژنتیک. در بخش 4-2-1 ما مقایسه میکنیم با روتینهای سورت دیگر و در

	AMD	Sun	SGI	IBM	IBM	Intel	Intel
CPU	Athlon MP	UltraSparcIII	R12000	Power3	Power4	Itanium 2	P4 Intel Xeon
Frequency	1.2GHz	750MHz	300MHz	375Mhz	1.3GHz	1.5GHz	3GHz
L1d/L1i Cache	128KB	64KB/32KB	32KB/32KB	64KB/64KB	32KB/64KB	16KB/16KB	8KB/12KB (1)
L2 Cache	256KB	1MB	4MB	8MB	1440KB	256KB (2)	512KB
Memory	1GB	4GB	1GB	8GB	32GB	8GB	2GB
OS	RedHat9	SunOS5.8	IRIX64 v6.5	AIX4.3	AIX5.1	RedHat7.2	RedHat8.2.3
Compiler	gcc3.2.2	Workshop cc 5.0	MIPSPro cc 7.3.0	Visual Age c v5	Visual Age c v6	gcc3.3.2	gcc3.4.1
Options	-O3	-native -xO5	-O3 -TARG: platform=IP30	-O3 -bmaxdata: 0x80000000	-O3 -bmaxdata: 0x80000000	-O3	-O3

Table 3: Test Platforms. (1) Intel Xeon has a 8KB trace cache instead of a L1 instruction cache. (2) Intel Itanium2 has a 6MB L3.

4-2-2 با کتابخانه های تجاری مثل INTEL MKL,C++ STL, IBM ESSL.

4-2-1 نتایج اجرا

ما استفاده میکنیم الگوریتمهای ژنتیک را برای تولید یک الگوریتم سورت برای سورت 32 بیت کلید اینتجر. الگوریتم میزان شده است با سورت نمودن مجموعه داد ورودی با اندازههای رنج شده ای از 8M برای 16M کلید و انحراف معیار رنج شده است از 2^9 تا 2^23.

برای ازمایشاتت در این بخش ما سورت میکنیم رکورده را با دو فیلد, 32 بیت کلید اینتجر و 32 بیت اشاره گر. ما استفاده میکنیم این ساختار را برای اینکه تا مینیمم کنیم حرکت داده را از رکوردهای طولانی نوعی از پایگاه داده ها, سورت کردن هست معمولا اجرا شده در یک ارایه از تاپلها, هر دربر داشتن یک کلید و اشاره گر برای رکورد اورجینال[15,18]. ما فرض میکنیم که این ارایه ساخته شده باشد قبل از روتینهای کتابخانه یمان نامیده شده اند.

شکل 7 نشان میدهد اجرا را از الگوریتمهای سورت مختلف: الگوریتم سورت QUICKSORT,CCRADIX,MULTIWAY MERGE,ADAPTIVESORT که ما ارائه شده در ,[14] و الگوریتم سورت تولید شده با استفاده از الگوریتم سورت (GENESORT). برای CC-RADIX سورت ما استفاده کردیم اماده کردن پیاده سازی را با تایفی از [11]. برای QUICKSORT,MULTIWAY MERGE,ADAPTIV سورت ما استفاده کردیم پیاده سازی که ما ارائه شده در [14]. QUICKSORT,MULTIWAY MERGE به صورت اتوماتیکی میزان شده بودند در هر پلتفرم معماری وار استفاده شده اند برای جستجوی تجربی برای شناسایی مقادیر پارامترها مثل گنجایش خروجی یا اندازه HEAP. الگوریتم ADAPTIVE سورت شده مجموعه داده با استفاده کردن الگوریتم خالص که ان پیشگویی میکند برای بهترین خروجی از سورت CC-RADIX,QUICKSORT,MULTIWAY MERGE سورت بحث شده در [14]. GENESORT هست الگوریتم تولید شده با استفاده از دیدگاه ارائه شده در این مقاله.

شکل 7 طرح کرده است زمان اجرا را در میکروثانیه((10^(-6)) با کلیدی مثل انحراف معیار متغیی بین 2^9 تا 2^23. ورودیهای تستس استفاده شده اند برای جمع اوری داده در شکل 7 شامل شده اند 14M رکورد و انحراف معیار از اندازه 4^N*512 با N رنج شده در 0 تا 8. این ورودیهای تست مختلف بودند از اینها استفاده شده در مدت انجام پردازش. برای هر انحراف معیار سه مجموعه ورودی مختلف با انحراف معیار یکسان سورت شده بودند با استفاده از 5 الگوریتم سورت مختلف. شکل PLOT میانگین از 3 زمان اجراست. تفاوت مابین این سه زمان اجرا کمتر از 3٪ میباشد. ورودیهای تستی دارند یک توزیع نرمال که توزیع شده بود از ورودیهای تربیت شده.

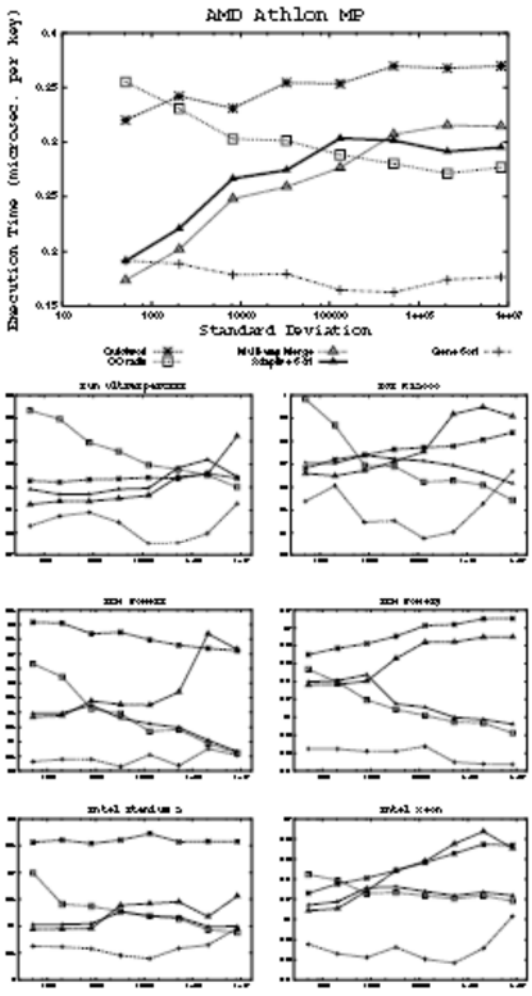


Figure 7: Performance of sorting algorithms as the standard deviation changes

به هر حال ما داریم اجرا میکنیم ازمایشات را که نشان میدهد که روتینهای سورت در شکل 7 تامین میکنند اجرایی شبیه را زمانی که ورودیها سورت شوند با توزیع تشریحی یا یکسان. این موافق است با نتایج که ما گزارش کردیم در [14].

ترجمه فا

شکل 7 نشان میدهد که GENESORT معمولا اجرا میکند بیشتر بهتر از CC-RADIX,MULTIWAY MERGE,QUICHSORT یا ADAPTIVE سورت الگوریتمم. الگوریتم ADAPTIVE ما ارائه شده است در یک پیش کار در [14] پیشگویی کردیم به صورت درست که بهترین الگوریتم در میان QUICKSORT,CC-RADIX,MULTIWAY MERGE میباشد اما این الگوریتمها معمولا اجرا میکنند بدتر از GENESORT ما و مثل نتیجه الگوریتم ADAPTIVE نمیتواند خارج کند اجرای GENESORT را. همچنین یادداشت که الگوریتم ADAPTIVE دارد بعضی سربار روی الگوریتم سورت پیشبینی شده به طوریکه ماشین پیشبینی شده نیاز دارد برای محاسبه انتروپی از مجموعه داده. انشعاب-با-انتروپی اولیه همچنین باعث میشود این سربار را اما ما خواهیم دید در بخش 4-3, در اخر هیچ یک از الگوریتمها پیدا نشده مثل با الگوریتم ژنتیک ما با استفاده از این اولیه.

اجرای GENESORT هست کمی بد (کمتر از 7٪) از بعضی از الگوریتمهای دیگر تنها در سه حالت: AMD ATHLON برای مقادیر بسیار کم از انحراف معیار و در SGI R12000 و INTEL ITANIUM2 برای مقادیر بسیار بالا از انحراف معیار. تابع FITNESS از الگوریتم ژنتیک ما در زمان جستجو کردن برای یک الگوریتم عمومی ترقی داده شده به الگوریتم سورت که میرسد به بهترین میانگین اجرا و نشان میدهد دگرگونی کم در زمان اجرا را (بخش 3-2-3). بنابراین استفاده کردن از این تابع FITNESS شاید نتیجه دهد در انتخابی از الگوریتم سورت با کمی رفتار بد برای انحرافمعیار بسیار بالا یا بسیار پایین به طوریکه انها نمایش میدهند مشخصات مختلف بسیار از بسیاری حالات. به هرحال نتایج ازمایشات ما نشان میدهد که الگوریتم GENESORT بسیار خوب اجرا میشود سراسر تمامی. همه جا در ATHLON MP که هست پلتفرم با مینیمم بهبود, الگوریتم سورت عمومی فراهم میکند یک مینگین بهبود 27٪ را. در دیگر پلتفرمها بهبود میانگین روی بهترین سه الگوریتم سورت خالص هست 35٪.

4-2-2 مقایسه اجرا با کتابخانههای تجاری

در این بخش ما مقایسه میکنیم اجرایی از GENESORT با روتینهای سورت از MATH KERNEL LIBRARY 7.0(MKL),C++ STL, IBM ESSL VERSION 3.2(IBM POWER3), VERSION 3.3(IBM POWER4)

زمانهای اجرا اندازه شده اند با ورودیهای سورت با 14M کلید. در IBM PLATFORM (POWER3 AND PAWER4)ها ما سورت کردیم 32 بیت کلید اینتجر را. در پلتفرمهای INTEL (INTANIUM 2AND XEON) ما سورت کردیم مقادیر ممیز شناور تک دقتی را به طوریکه INTEL MKL سورت نکرد اینتجرها را. ما میتوانیم بکار ببریم RADIX برمبنای اولیه ها را برای سورت کردن مقادیر ممیز شناور برای اینکه استفاده کردیم IEEE 754 استاندارد را ترتیب مرتبط از دو عدد ممیز شناور غیر منفی هست یکسان مثل ترتیبی از رشته بیتها که ارائه کردیم انها را[9]. کلیدها برای سورت قرار کرفته اند در مقعیتهای پیاپی در یک ارایه. برای

ازمایشات در این بخش ما شامل نمیکنیم اشاره گرهای استفاده شده در بخش قبلی را. ما دوباره تولید میکنیم کتابخانه های سورت را برای گرفتن داخل اکانت متفاوت (عداد ممیز شناور برای INTEL و غیر اشاره گر ها).
شکل 8 نشان میدهد زمان اجرا را از INTEL MKL,C++ STL, IBM ESSL , GENESORT, روتینهای سورتشان (خطمش متناظر به XSORT بحث خواهد شد در بخش بعدی). برای پلتفرم INTELپلیتفرم ما نشان دادیم همچنین QUİCKSORT را بهطوریکه INTEL MKL انجام داده است QUICK SORT را. برای ساده کردن پلوتها ما نشان ندادیم نتایج را برای دیگر روتینهای سورت در شکل 7 اما GENESORT همیشه اجرا میکند بهتر از هر کدام انها.
GENESORT هست سریعتر از C++STL در هر دوی IBM POWER3 و POWER4. در IBM POWER4 سورت GENESORT هست بسیار سریعتر از روتین سورت IBM ESSL. به هرحال در IBM POWER3,IBM ESSL روتین سورت اجرا میکند سریعتر از GENESORT. ان قابل توجه است که IBM ESSL روتین سورتش نیاز دارد به چرخه بسیار در POWER4 از POWER3(170 در مقابل 90 چرخه کلید). یک توضیح ممکن هست اینکه کتابخانه IBM ESSL بود بهطور دستی میزان شده برای POWER3 و نیست برای POWER4. اگر فرض ما هست درست این میبایستی نشان دهد معایب را از میزان سازی دستی در مقابل میزان سازی اتوماتیک با استفاده کردن دیدگاهمان. GENESORT تشکر میکند با میزان سازی اتوماتیک اجرای یکسان در هر دوی پلتفرمها اگر چه ان هست بیرون اجرا شده با کتابخانه IBM ESSL در POWER3. در بخش 5 ما ارائه کردیم یک دیدگاه کمی مختلف که تولید میکند روتین XSORT را. همچنین میتوان دید در شکل 8 XSORT هست سرعتر از هر کدام از کتابخانه های تجاری که ما رسیدگی کردیم(شامل کتابخانه IBM ESSL در POWER3). در میانگین XSORT هست 26٪ و 62٪ سریعتر از IBM ESSL در POWER3 و POWER4 به ترتیب.
در INTEL ITANIUM2 و QUICKSORT XEONمان که بهینه بود با استفاده جستجوی تجربی هست سریعتر از INTEL MLK در هر دو پلتفرم. C++STL هست بهصورت مرزی کند تر از QUICKSORT ما در بیشتر نقاط در INTEL XEON اما سریعتر از QUICKSORTما در INTEL ITANIUM2. به هر حال C++STL ISJ FSDHV HISJI JV HC GENESORT ما در هر دو پلتفرم. XSORT اجرا میشود در میانگین 56٪ و 61٪ سریعتر از C++STL در INTEL ITANIUM2 و INTEL XEON به ترتیب.

3-4 انالیز بهترین سورت کردن ژنتیکی

جدول 4 ارائه میکند بهترین سورت ژنتیکی پیدا شده در ازمایشات بخش 4-2-1 برای روتینهای GENESORT در شکل 7. ارائه سورت از ژنتیک هست در فرم (پارامترهای اولیه(CHILD2)(CHILD1)...), که پارامترها هستند انهایی که نشان داده شده اند در جدول 1. برای مثال ((20 13 ldr)19 drوسیله بکار میبرد

radix سورت را با radix19 و سپس radix سورت با یک radix13 و threshold 20 برای بکار بردن در سورت in-place-register(ببینید بخش 2).

یک انالیز از نتایج در جدول 4 , راهنمایی به ما برای دو مشاهد اصلی: 1) radix مبنا شده بر اساس اولیه های تقسیم-با-radix و —dive‌با—radix-بافرضنمودن-یکنواختی توزیع (DU) هستند بسیار فرکانسی استفاده شده اولیه. در ابتدا داده هستند پارتیشن بندی شده با استفاده از RADIX سورت با یک RADIX بزرگ (15 یا بزرگتر). بنابراین انها هستند پارتیشن بندی شده دوباره با RADIX سورت با RADIX کمتر. 2) انتخاب اولیه ها هستند کمیاب و تنها انشعاب-با-اندازه (BS) اولیه پدیدار شده در IBM POWER4 و INTEL XEON.

AMD Athlon MP	Sun UltraSparc III	SGI R12000	IBM Power3	IBM Power4	Intel Itanium 2	Intel Xeon
(dr 15(dr 9 (ldr 5 20)))	(dr 19(du 6 (ldr 7 20)))	(dr 17(dr 6 (ldr 9 5)))	(dr 19(ldr 13 20))	(dr 16(bs 1186587 (ldr 5 20) (du 3(ldr 13 20))))	(dp 246411 4 (ldr 5 20))	(dr 17 (bs 1048576 (ldr 5 20) (du 6 (ldr 9 20))))

Table 4: Gene Sort algorithm for each platform.

انشعاب-با-انتروپی(BE) اولیه پدیدار نشد. ما توانا نیستیم برای بازبینی اینکه سورت ژنتیک در جدول 4 هستند بهینه برای هر پلتفرم بهطوریکه فضای جستجو هست بقدری بزرگ که جستجوی وسیع نیست ممکن. به هر حال ما هدایت کردیم بعضی آزمایشات گزارش شده را بعدا, برای وارسی کردن بهینگی الگوریتم پیدا شده.

ما داشتیم یک مطالعه حسساس برای رسیدگی که پارامترها پیدا شده هستند بهینه. ما گرفته بودیم سورت ژنتیکی را برای AMD ATHLON MP و IBM POWER3 در جدول 4 و ما تغییر دادیم انها را با تغییر اندازه RADIX. زمانی که تغییر دادن اندازه RADIX از RADIX ابتدایی انجام شد نتایج ما نشان میدهد که اندازه RADIX انتخاب شده با الگوریتم ژنتیک ما هست یکی از سریعترین اجرا, اگرچه ان شاید بعضی وقتها اتفاق میافتد کش وافر یا TLB فقد شده. آزمایشات ثابت کردند RADIX ابتدایی و تغییر دادن مقدار از RADIX دوم همچنین نشان داده شده که پارامتر انتخاب شده با الگوریتم ژنتیک ما برای این RADIX دوم واقعا بهترین بود. پس ان پدیدار شده که دیدگاه ما به طور موثر پیدا میکند بهترین مقادیر پارامتر در کمترین برای الگوریتمهای مبنای RADIX و پلتفرم که ما امتحان کردیم.

ملاحظه کنید که اولیه های انتخاب هستند کمیاب در الگوریتمهای سورت اشاره میکنند که الگوریتم ژنتیک ما تولید میکند کدی که هست ناتوان برای تبدیل به انحراف معیار از مجموعه داده ورودی. برای مطالعه کردن چگونگه اجرا میتواند بهبود یابد اگر کد تولید شده میتواند داشته باشد اینچنین تبدیل را, ما تغییر میدهیم سیستممان را برای تولید یک سیستم دسته کننده.

5 - دسته کننده سورت

در این بخش ما خارج میکنیم چطوری برای استفاده کردن الگوریتمهای ژنتیک برای تولید یک سیستم دسته کنند [3,16,22] برای روتینهای سورت که هستند درخواست شده برای ناحیه های مختلف از فضای ورودی. زمانی که تولید نمودن یک دسته کننده انتخاب از یک ژنتیکی در یک ناحیه ای از فضای ورودی وابسته نباشد به اجرایی از ژنتیکی در یک ناحیه متفاوت.

یک سیستم دسته کننده ترکیب شده است از یک مجموعه قوانین. هر قانون دارد دو قسمت یکی شرط و یکی کنش. شرط هست یک رشته که انکد میکند مشخصات امین را از ورودی که هر عنصر از رشته میتواند داشته باشد سه مقدار ممکن: (DON'T CARE)*,1 , 0. به طور مشابه مشخصات ورودی انکد میشوند با یک بیت رشته از طول یکسان. اگر i و c هستند ورودی رشته بیت و شرط رشته به ترتیب, ما میتوانیم تعریف کنیم تابع MATCH(p,c) مثل زیر:

$$match(i,c) = \begin{cases} true, & \forall(j) i_j = c_j \vee c_j =' *' \\ false, & otherwise \end{cases}$$

اگر match(i,c) هست true , کنش متناظر برای شرط بیت رشته c انتخاب خواهد شد. یک پیشگویی fitness هست اختصاص داده شده به هر قانون.

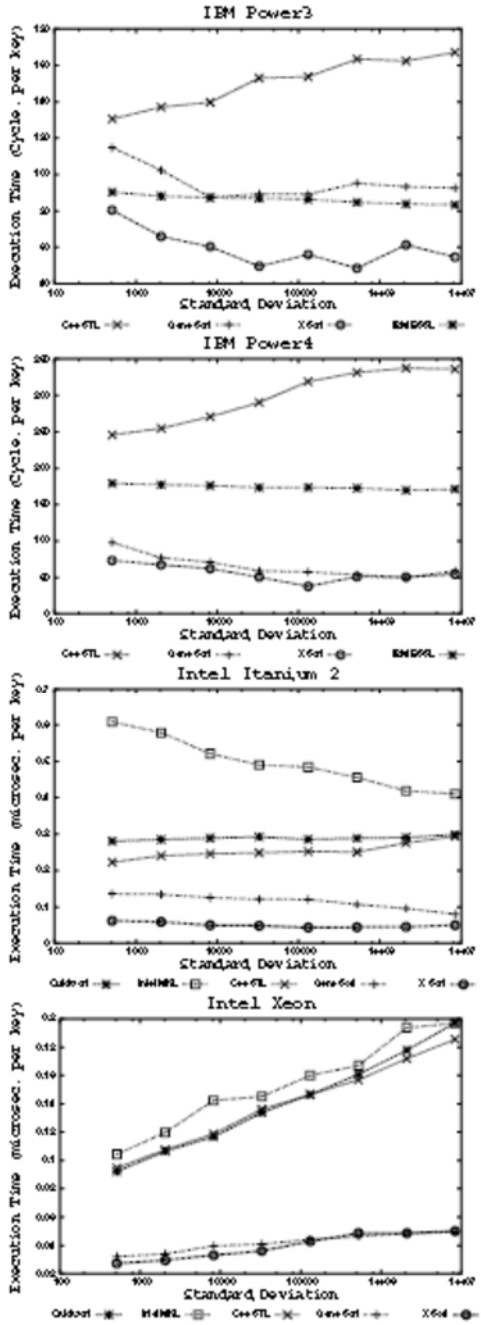


Figure 8: Performance comparison with commercial libraries.

برای یک رشته ورودی داده شده اینجا میتواند باشد قوانین مچ شده چند گانه. قانون با بالاترین پیشگویی fitness انتخاب خواهد شد با کنش در ورودی. با * در رشته شرط دو یا بیشتر رشته ورودی میتواند مشترک شود با کنش یکسان. بعدا ما مرور میکنیم چطوری سیستم دسته کننده میزان شده برای هر پلتفرم و ورودی.

5-1 ارائه

همان طور که ما مرور کردیم در [14] و مطرح کردیم در بخش 2

ترجمه فا

اجرا از سورت وابسته است به تعداد کلیدها N و انتروپی از هر عدد Ei. بنابراین رشته بیت شرط دارد برای انکد کردن مقادیر مختلف از این دو مشخصات ورودی. تعداد بیتها استفاده میشود برای انکد کردن مشخصات ورودی در رشته بیت شرط وابسته خواهد بود به شدت در اجرایی از هر پارامتر ورودی. پس بیشتر شدت و پیچیدگی یک پارامتر ورودی دارد در اجرای بالاتر تعداد بیتهایی که میبایستی استفاده شده باشند برای انکد کردن این پارامتر ورودی.

نتایج آزمایشات ما نشان میدهد که انتروپی دارد یک پیچیدگی بالا تر در اجرایی نسبت به تعداد کلیدها.مثل نتایج ما تصمصم گرفتیم برای استفاده دو بیت برای انکد تعداد کلیدها و چهار بیت برای انکد انتروپی از هر عدد. بنابراین اکر ما فرض کنیم که N میتواند رنج کند از 4M تا 6M, , انکد کردن دیفرنسیالی ما بین چهار ناحیه از طول 3M هر کدام.

انتخاب الگوریتم هست انجام شده با استفاده از مکانیسم مچ کردن. مثل نتایج اولیه های انتخاب نیستند طولانی نیاز دارند برای انتخاب اولیه های سورت مناسب. بنابراین قسمت عمل از یک قانون تنها مرکب است از نقشه های سورت با اولیهای انتخاب ما. سورت ژنتیکی الان دارد فرمی از یک لیست خطی نیست درخت.

دادن یک ورودی به سورت , مشخصات ورودییشان N و E انکد خواهد شد داخل رشته بیت i . تمام شرطهای Cj در مجموعه قانون از سیستم دسته بندی مقایسه خواهد شد دوباره رشته بیت ورودی i. تمام شرطها مچ میکنند رشته بیت ورودی i ترکیب شده مجموعه مچ M.

5.2 تربیت کردن

ما تربیت میکنیم سیستم دسته کننده را برای یاد گرفتن یک مجموعه از قوانین که پوشش میدهند فضایی از مقادیر پارامتر ورودی ممکن را, پیدا کردن شرایط که بهترین تقسیم را از فضای ورودی و میزان کردن کنشها برای یادگیری بهترین ژنها برای سورت ورودیها با مشخصات شناخته شده در شرط. همچنین قبل از مدتی که تربیت کردن پردازش میکند ورودیها را با مقادیر مختلف از E AND N تولید شده هستند و سورت شده.

گرفتن یک ورودی تربیت شده , ما داریم یک مجموعه قانون مچ شده, به طوریکه هستند مجموعه ای از قوانین که شرط مچ میکند(مطابقت میدهد) رشته بیت را برای انکد کردن مشخصات ورودی. ما میتوانیم تولید کنیم قوانین مچ کردن تازه را برای بکار بردن دگرگون سازی هر دوی رشته شرط و کنش مثل انچه که توضیح داده شد در بخش 3-2-2.

5-3 زمان اجرا

در پایان فاز تربیت کردن , ما یک مجموعه قانون داریم. در زمان اجرا رشته بیت انکد میکند مشخصات ورودی را که استفاده خواهند شد برای استخراج مجموعه مچ. بیرون از این قانون ها , یکی با بالاترین پیشگویی اجرا و دقت

انتخاب خواهند شد, و عمل متناظر خواهد بود الگوریتم سورت استفاده میکند برای سورت ورودی. سربار زمان اجرا شامل میشود با محاسبه انتروپی برای انکد رشته بیت ورودی و اسکن مجموعه قانون برای انتخاب بهترین قانون. در در انتروپی آزمایشات ما محاسبه شده است با نمونه برداری یکی از چهار کلید از ورودی. این سربار جزیی محاسبه شده است برای زمان نیاز شده برای سورت ارایه های ورودی از اندازه (>=4M).

4-5 نتایج آزمایشات

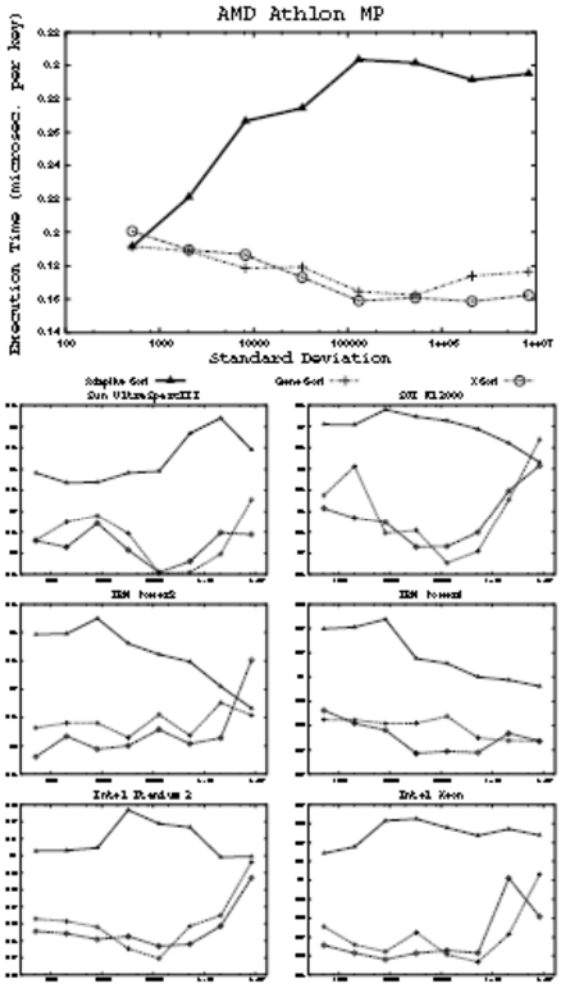


Figure 9: Xsort versus Gene and Adaptive Sort.

ترجمه فا

	8192	524288	8388608
Sun UltraSparc III	(dr 19 (dr 13))	(dr 21 (ldr 5 20))	(dr 21 (du 5 (ldr 6 20)))
IBM Power3	(dr 22 (du 10))	(dr 19 (du 7 (ldr 6 20)))	(dr 20 (du 10 (ldr 2 20)))

Table 5: Best genomes selected by the classifier sorting library.

شکل 9 مقایسه میکند زمان اجرا را از الگوریتمهای تولید شده در فرمی از یک کلاس کننده (Xsort) در مقابل genesort و سورت adaptive مثل تغییر انحراف معیار زمانی که سورت هست 14M رکورد. Xsort هست تقریبا همیشه بهتراز سورت adaptive . در معدل Xsort هست 9٪سریعتر از genesort, شروع شدن با 12٪ سریعتر از genesort در IBM POWER4. زمانی که مقایسه شد با سورت ADAPTIVE (که هست مقایسه شده از الگوریتمهای سورت خالص) Xsort در معدل 36٪ سریعتر است با شروع شدن 45٪ سریعتر در intel xeon.

جدول 5 نشان میدهد تفاوت سورت کردن ژنتیکی ژیدا شده را با ساتفاده کلاس کننده برای 14M کلید و مقادیر متفاوت از انحراف معیار برای SUN ULTRASPARC III و IBM POWER 3. جدول نشان میدهد که الگوریتمها هنوز هستند بر مبنای RADIX اما انها متفاوت هستند بر مبنای انتروپی.

6 نتیجه

در این مقاله ما ساختن الگوریتمهای سورت ترکیبی را از اولیه ها برای تبدیل به پلتفرم هدف و داده ورودی پیشنهاد کردیم. الگوریتمهای ژنتیک برای جستجو برای روتینهای سورت استفاده شدند. نتایج ما نشان میدهد که بهترین روتینهای سورت فراهم شده اند زمانی که الگوریتمهای ژنتیک برای تولید یک سیستم کلاس کننده استفاده شوند. الگوریتم نتیجه شده یک الگوریتم ترکیبی میباشد که یک روتین سورت متفاوت بر مبنای انتروپی و تعداد کلیدها برای سورت انتخاب شده است. دربیشتر حالات روتینها RADIX بر مبنای پارامترهای متفاوت وابسته شده در مشخصات ورودی و ماشین هدف میباشند. الگوریتم سورت تولید شده هست در معدل 36٪ سریعتر از بهترین روتین سورت خالص در 7 پلتفرم که ما ازمایش کردیم برای شروع 42٪ سریعتر میباشد. روتین تولید شده ما بهتر از هر روتین ترکیبی که ما امتحان کردیم در IBM ESSL, INTEL MKL و STL از C++ اجرا میکند. در میانگین روتین تولید شده ما 26٪ و 62٪ سریعتر از IBM ESSL در یک IBM POWER 3 و IBM POWER4 به ترتیب میباشد.

ترجمه فا

منابع

References

[1] T. Back, D. B. Fogel, and Z. Michalewicz. *Evolutionary Computation Vol. I & II*. Institute of Physics Publishing, 2000.

[2] J. Bilmes, K. Asanovic, C. Chin, and J. Demmel. Optimizing Matrix Multiply using PHiPAC: A Portable, High-Performance, ANSI C Coding Methodology. In *Proc.of the 11th ACM International Conference on Supercomputing (ICS)*, July 1997.

[3] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. *Lecture Notes in Computer Science*, 1996:253+, 2001.

[4] K. Cooper, P. Schielke, and D. Subramanian. Optmizing for Reduced Code Space Using Genetic Algorithms. In *Proc. of the Workshop on Languages, Compilers and Tools for Embedded Systems*, pages 1–9, May 1999.

[5] J. Darlington. A Synthesis of Several Sorting Algorithms. *Acta Informatica*, 11:1–30, 1978.

[6] P. K. et al. Finding Effective Optimization Phase Sequences. In *Proc. of the Conf. on Languages, Compilers and Tools for Embedded Systems*, pages 12–23, June 2003.

[7] M. Frigo. A Fast Fourier Transform Compiler. In *Proc. of Programing Language Design and Implementation*, 1999.

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[9] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[10] C. Hoare. Quicksort. *Computer*, 5(4):10–15, April 1962.

[11] D. Jim ´ enez-Gonz ´ alez, J. Navarro, and J. Larriba-Pey. CC-Radix: A Cache Conscious Sorting Based on Radix Sort. In *Euromicro Conference on Parallel Distributed and Network based Processing*, pages 101–108, February 2003.

[12] H. Johnson and C. Burrus. The Design of Optimal DFT Algorithms Using Dynamic Programming. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31:378–387, April 1983.

[13] D. Knuth. *The Art of Computer Programming; Volume3/Sorting and Searching*. Addison-Wesley, 1973.

[14] X. Li, M. J. Garzar ´ an, and D. Padua. A Dynamically Tuned Sorting Library. In *In Proc. of the Int. Symp. on Code Generation and Optimization*, pages 111–124, 2004.

[15] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. AlphaSort: A RISC Machine Sort. In *Proc. of the Sigmod Conference*, pages 233–242, 1994.

ترجمه فا

[16] W. S. Pier Luca Lanzi and S.W.Wilson. *Learning Classifier Systems, From Foundations to Applications*. Springer-Verlag, 2000.
[17] R. Sedgewick. Implementing Quicksort Programs. *Communications of the ACM*, 21(10):847–857, October 1978.
[18] A. Shatdal, C. kant, and J. Naughton. Cache Conscious Algorithms for Relational Query Processing. In *Proc. of the 20th Int. Conference on Very Large Databases*, pages 510–521, 1994.
[19] B. Singer and M. Veloso. Stochastic Search for Signal Processing Algorithm Optimization. In *Proc. of Supercomputig*, 2001.
[20] M. Stepehnson, S. Amarasinghe, M.Martin, and U. O'Reilly. Meta Optimiization: Improving Compiler Heuristics with Machine Learning. In *Proc. of Programing Language Design and Implementation*, June 2003.
[21] R. Whaley, A. Petitet, and J. Dongarra. Automated Empirical Optimizations of Sofware and the ATLAS Project. *Parallel Computing*, 27(1-2):3–35, 2001.
[22] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
[23] J. Xiong, J. Johnson, R. Johnson, and D. Padua. SPL: A Language and a Compiler for DSP Algorithms. In *Proc. of the International Conference on Programming Language Design and Implementation*, pages 298–308, 2001