Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Papler

Algoritemsko nadzorovano trgovanje z vrednostnimi papirji

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

prof. dr. Miha Mraz
MENTOR

Ljubljana, 2014

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00571 / 2013
Datum: 13.12.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ PAPLER**

Naslov: **ALGORITEMSKO NADZOROVANO TRGOVANJE Z VREDNOSTNIMI PAPIRJI**
ALGORITHMICALLY CONTROLLED TRADING OF SECURITIES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Uporaba računalniških tehnologij ima v današnjem času velik pomen pri trgovanju z različnimi vrednostnimi papirji. Kandidat naj v svojem delu predstavi osnove algoritemskega trgovanja, metode trgovanja ter dostopne programske platforme, s katerimi ga lahko izvajamo. Na izbrani programski platformi MT4 naj identificira pomanjkljivosti, ter vsaj eno od njih tudi odpravi s programsko nadgradnjo sistema. Delovanje dodatnih funkcionalnosti naj predstavi z enostavnim odločitvenim algoritmom.

Mentor:

prof. dr. Miha Mraz

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela, da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih kjer, so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom prof. dr. Mihe Mraza,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko in
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Matej Papler, Ljubljana, marec 2014.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Papler

Algoritemsko nadzorovano trgovanje z vrednostnimi papirji

POVZETEK

V informacijski dobi elektronsko trgovanje z vrednostnimi papirji in valutami ni le možnost, ampak pogosto edini način trgovanja. Podatke iz trga je mogoče obravnavati z algoritmi, ki so narejeni z namenom avtomatskega trgovanja, torej da lahko algoritmi brez človeškega posredovanja kupujejo in prodajajo vrednostne papirje na trgu.

Diplomska naloga v prvem delu predstavi metode algoritemskega trgovanja, prednosti in slabosti ter orodja s katerimi ga lahko izvajamo. V drugem delu predstavi nadgradnjo programskega sistema MetaTrader4, ki odpravi dve njegove pomanjkljivosti: dodana je možnost sinhrone simulacije trgovalnih algoritmov ter izvajanje poljubnega trgovalnega algoritma sočasno na redundančnih računalnikih. Za prikaz delovanja dodanih funkcionalnosti je podan tudi primer enostavnega trgovalnega algoritma.

Kljucne besede: algoritemsko trgovanje, Meta Trader, MQL, sinhronizirana simulacija

University of Ljubljana
Faculty of Computer and Information Science

Matej Papler

Algorithmically controlled trading of securities

ABSTRACT

Electronic trading of securities in the information age is no longer just one of the options but often the only one. Data from the market can be processed in algorithms that are designed for the purpose of automated trading, meaning that such algorithms can buy and sell securities on the market without human intervention.

This thesis in it's first part describes the methods of algorithmic trading, advantages and disadvantages, and the available tools that make it possible. In the second part it describes an upgrade to the MetaTrader4 program system that solves two of it's deficiencies: ability to make a synchronous simulation of trading algorithms that are dependent of eachother; and to allow a trading algorithm to run concurrently on multiple computers to provide redundancy. A sample trading strateg was created to demonstrate the added functionality.

Key words: algorithmic trading, Meta Trader, MQL, synchronized simulation

ZAHVALA

Zahvaljujem se staršema za podporo in potrpežljivost tekom številnih let študija. Poloni, ki mi je stala ob strani. Borutu za mnoga dodatna znanja. Tistim, od katerih sem povzel življenjske vrednote. Ter uporabnikom mojih stvaritev.

Še posebej se zahvaljujem prof. dr. Mihi Mrazu, za izjemno hitre popravke mnogoterih slovničnih napak in za vse nasvete pri izdelavi te diplomske naloge.

— Matej Papler, Ljubljana, marec 2014.

KAZALO

1 Uvod

Bistvo vsakega trgovanja je opravljati posle in z njimi dosegati dobiček. Pri trgovanju z vrednostnimi papirji, valutami in zamenljivim blagom (angl. *fungible goods*), torej ko ne gre za kvaliteto ampak govorimo samo o kvantiteti, se koncept trgovanja bistveno poenostavi: opravka imamo samo s ceno in količino. Analiza zgodovine cen in količin ter s tem napovedovanje cene v prihodnosti je prisotna že od začetka borz. S prihodom računalnikov pa uporaba matematičnih ved in posledično računanja ni več pomožno orodje, ampak je postala osnovno orodje tistih, ki na trgovanje gledajo iz tehničnega vidika. Z informatizacijo borz v 70. letih 20. stoletja izvajanje poslov na borzi ne zahteva več papirja, temveč se vse izvaja elektronsko. S tem je bila odprta možnost, da računalnik ni le pomožno orodje, ampak je mogoče izdelati algoritem, ki analizira trg ter z določenimi pravili sam izvaja naročila.

V diplomski nalogi poizkušamo bralca seznaniti z algoritemskim trgovanjem – njegovo zgodivino, kako in zakaj se uporablja. Opisane so prednosti, ki jih ponuja, kot tudi slabosti, katere moramo upoštevati pri takem trgovanju. Predstavljenih je nekaj bolj popularnih programskih rešitev, namenjenih končnim uporabnikom.

V drugem, bolj praktičnem delu, smo se osredotočili na nadgradnjo programskega paketa MT4 (Meta Trader 4) z namenom odprave nekaterih pomanjkljivosti. Identificirali smo dva problema. Prvi problem predstavlja nezmožnost MT4, da bi na preteklih podatkih različnih vrednostnih papirjev simulirali algoritme, ki se izvajajo sočasno in so medsebojno odvisni. Drugi problem predstavlja potreba po redundančnosti, torej zmožnosti, da se izvajanje neke trgovalne strategije nadaljuje tudi ob izpadu enega ali več računalnikov, na katerih teče algoritem. Pri izdelavi te nadgradnje smo poleg dodatnih funkcionalnosti hoteli tudi čimbolj poenostaviti delo tistim uporabnikom, ki želijo le izdelati trgovalni algoritem in se ne želijo posvečati sistemskim vidikom delovanja platforme MT4 oziroma naših dodatnih funkcionalnosti.

Za konec smo izdelali tudi enostaven algoritem, s katerim prikažemo delovanje dodanih funkcionalnosti sistema.

Pričujoča diplomska naloga v poglavju 2 opiše algoritemsko oziroma avtomatsko trgovanje, prednosti in slabosti le-tega, ter programske sisteme, ki ga omogočajo. V poglavju 3 predstavimo razširitev programskega sistema MetaTrader 4, ki naj bi odpravila dve ključni pomanjkljivosti tega sistema, zaradi katerih ga težko uporabljamo za resno trgovanje:

(I) realiziramo sistem simulacije algoritmov, ki se izvajajo sočasno na različnih vrednostnih papirjih in se med seboj povezujejo,

(II) omogočimo redundačnost, torej da lahko poljubni algoritem teče sočasno na več lokacijah, s čimer dosežemo večjo robustnost v primeru odpovedi enega od računalniških sistemov.

V poglavju 4 je opisano delovanje vzorčnega algoritma za trgovanje, napisanega v programskem jeziku MT4, ki uporablja dodane funkcionalnosti iz poglavja 3.

2 Algoritemsko trgovanje

V modernem svetu vrednostni papirji skorajda ne obstajajo več izven računalnika. Na primer: vezana vloga na banki je vrednostni papir. Enako velja za kreditno pogodbo. Vrednosti teh pogodb, torej denar, ki jih predstavljajo, nato skupaj poveže računalnik. Pogosto neka banka nima enake vrednosti kreditov in depozitov, ter to razliko ponudi drugim bankam na različnih borzah[1]. Trguje se tudi z vrsto drugih vrednostnih papirjev oziroma pogodb. Te vključujejo pogodbe za dobavo surovin, kot so nafta in koruza, do dragih kovin ter pogodb za omejitve izpustov ogljikovega dioksida. Pomemben del prometa z vrednostnimi papirji predstavljajo delnice in obveznice podjetij ter državne obveznice.

Borza naročila za posle prejema preko računalniškega omrežja. Torej kupec oziroma prodajalec nekega vrednostnega papirja je lahko človek, ki preko računalnika odda naročilo, lahko pa je tudi računalnik sam. Pravzaprav je v tem primeru kupec algoritem, narejen z namenom opravljanja poslov, bodisi da človeku olajša delo ali pa trguje na podlagi izračunov, ki so prezahtevni, da bi jih človek lahko opravil ročno v dovolj hitrem

[1] Za valuto Evro se uporablja Euribor (angl. *Euro Interbank Offered Rate*).

času.

Namen trgovanja je doseganje čim večjih dobičkov, pri tem pa je potrebno kar se da zmanjšati tveganje. Odločanje o sklepanju poslov se v splošnem deli na dve vrsti:

- Fundamentalna analiza [1] (angl. *fundamental analysis*) - kriteriji so makroekonomski, ki jih ni mogoče opisati v številkah (npr. stanje in pričakovanja v ekonomiji, politiki, tehnologiji, vremenu, itd.),
- Tehnična analiza [2] (angl. *technical analysis*) - zanašamo se na natančno merljive podatke (npr. trenutna in pretekla cena instrumenta, podatki o obrestnih merah, likvidnosti, kapitalizacije, itd.).

Pri algoritemskem trgovanju gre vedno za odločanje na podlagi tehnične analize, torej glede na številske podatke, ki jih je mogoče točno določiti in na njihovi podlagi odločati o sklepanju poslov.

2.1 Zgodovina trgovanja z vrednostnimi papirji

Začetki trgovanja z vrednostnimi papirji segajo v antični Rim, kjer so ljudje trgovali s pogodbami za izgradnjo objektov za državo. V 12. stoletju je Beneška Republika izdala državne obveznice, ki so bile prenosljive, torej so ljudje z njimi lahko trgovali. Leta 1602 je podjetje Dutch East India Company iz Nizozemske postalo prvo, ki je izdalo delnice. Za elektronsko trgovanje pomemben mejnik predstavlja odprtje elektronske borze NASDAQ[2] leta 1971. Tako se je trgovanje s fizičnimi papirji spremenilo v trgovanje preko računalnika. Sprva so bili vsi posli ročno vneseni v sistem, potem pa se je tekom desetletja razvil trend računalniško generiranih poslov.

V Združenih državah Amerike so bile prvotno vse cene vrednostnih papirjev na borzah izražene kot ulomek, na primer 37 in $\frac{1}{4}$ dolarja. Leta 2000 pa so se začele pojavljati cene izražene v decimalnem sistemu, na primer 37,03 dolarja. Regulatorna agencija je nato določila, da morajo biti vse cene izražene na decimalen način, najkasneje do aprila leta 2001. To naj bi vzpodbudilo trgovanje, tako ročno kot algoritemsko, zaradi veliko pogostejših sprememb v ceni. Tako povečan promet trgovanja je pomenil tudi večjo likvidnost trgov.

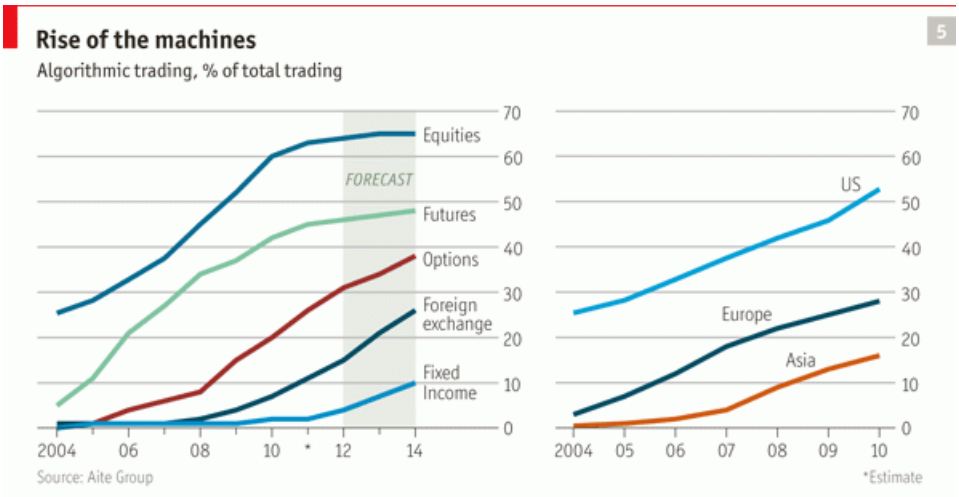
V letu 2001 so raziskovalci podjetja IBM objavili raziskavo [3], ki prikazuje, da lahko algoritem trguje bolj dobičkonosno kot človek.

[2] National Association of Securities Dealers Automated Quotations

2.2 Prednosti algoritemskega trgovanja

Pozorno sledenje spremembam na trgu se lahko izkaže zahtevno za človeka, to nalogo pa lahko v določenih primerih prepustimo računalniku. Poleg tega, da se računalnik ne utrudi, lahko določeno strategijo testiramo na zgodovinskih podatkih hitreje, enostavneje, ceneje in bolj zanesljivo, kot če bi dobičkonosnost in tveganje ročno izračunal človek. Zaradi enostavnosti testiranja lahko strategijo testiramo s spremenjenimi parametri delovanja, da ugotovimo optimalnejše vrednosti parametrov.

Delež poslov, ki izvirajo iz algoritemskih metod, danes predstavlja večino v nekaterih segmentih elektronskega trgovanja. Trend rastoče popularnosti je prikazan na sliki 2.1.



Slika 2.1 Odstotek algoritemskega trgovanja v obdobju med letoma 2004 in 2012, ter napoved za leti 2013 in 2014 [4].

2.2.1 Konsistentnost

Ročno trgovanje z vrednostnimi papirji zahteva zbranost in čas. Trgovalci, ki trgujejo po neki trgovalni strategiji, se jo morajo držati do vsake podrobnosti. Ves čas morajo disciplinirano slediti strategiji. Pogosto se lahko zgodi, da se trgovalec odloči drugače zaradi faktorjev, ki sicer izgledajo pomembni, toda kasneje se izkaže, da je šlo samo za zamegljen razum.

Primer pogoste napake ročnega trgovanja je, da ko je nek posel dobičkonosen, ga trgovalec ne zaključi kljub pravilom strategije, ampak upa na še večji dobiček, kar pa lahko vodi v končanje posla z manjšim dobičkom, kot je predvideno glede na analizo preteklih situacij. Podobna situacija nastane, kadar ima posel izgubo - trgovalec z njim

vztraja v upanju, da se bo izguba spremenila v dobiček, kljub temu da to lahko vodi v še večjo izgubo. Poleg tega prihaja do obdobij, ko daljši čas ni pogojev za odprtje nobenega posla, ter si trgovalec začasno spremeni pogoje vstopa v posel.

Računalniški program se ne sooča s takimi dilemami in v vsak posel vstopi in iz njega izstopi točno po natančno določenih pravilih brez izjem.

2.2.2 Časovne omejitve

Trgovanje na nekaterih borzah je omejeno na določen čas v dnevu[3] kar časovno določa zmožnost izvajanja naročil, nekatere pa so odprte neprekinjeno[4]. Pri ročnem trgovanju moramo upoštevati človeške potrebe (spanje, hrana, toaleta), zato je težko spremljati spremembe na trgu ves čas aktivnega delovanja borze. Problem delne človeške prisotnosti lahko rešujemo na več načinov. Na primer, da strategijo določimo tako, da le v določenih urah v dnevu iščemo vstop v posel, ta pa je odprt največ nekaj ur. Za zaprtje posla lahko uporabimo tudi omejitve izgube in dobička (angl. *take profit & stop loss*), tako da za njegovo zaprtje poskrbi trgovalni strežnik. En sam trgovalec pa težko izvaja strategijo, ki predvideva vstop ali izstop samo glede na pogoje na trgu, torej kadar je čas vstopa ali izstopa nepredvidljiv. To pomeni, da mora trgovalec aktivno spremljati trg ves čas, ko je odprt. Za izvedbo poslov lahko v tem primeru poskrbimo tako, da posle izvaja več ljudi v izmenah. V nekaterih primerih je enostavnejša in cenejša rešitev izdelava ustreznega algoritma.

2.2.3 Testiranje strategije na obstoječih podatkih

Trgovalni algoritem lahko izvajamo s testnim orodjem, s čimer lahko preizkusimo uspešnost njegovega delovanja. Testno orodje lahko algoritmu podaja zgodovinske podatke nekega vrednostnega papirja, s čimer ugotovimo dobičkonosnost in tveganje tega algoritma. Glede na rezultate v preteklosti lahko sklepamo o uporabnosti neke strategije v prihodnosti.

2.2.4 Optimizacija trgovalne strategije

Pogosto se pri izdelavi strategije za trgovanje pojavijo potrebe po upoštevanju dodatnih podatkov, spremembe v izračunih pogojev vstopa in izstopa, ter prilagoditev parametrov

[3] Trgovalne ure na borzi NASDAQ so vsak delovnik med 9:30 in 16:00, na nemški platformi XETRA pa med 9:00 in 17:30.

[4] Valutni trg (angl. *foreign exchange* oziroma FOREX), odprt 24 ur na dan vse delovne dni.

za bolj optimalno trgovanje. Analiza takih sprememb je za človeka časovno zahtevna, ter pravzaprav nemogoča, če gre za večjo količino podatkov, kar pa ne predstavlja bistvenega problema za računalnik. V kolikor imamo vse podatke, s katerimi strategija deluje, je mogoče na teh podatkih simulirati strategijo in s tem določiti njeno dobičkonosnost in tveganje. Ročna simulacija je lahko zelo zamudna, z ustreznimi simulatorji pa je lahko narejena v veliko krajšem času, poleg tega pa lahko izključimo tudi možnost človeške napake.

2.2.5 Dostopnost

V začetkih algoritemskega trgovanja je bilo le to domena velikih institucij, kot so banke in vzajemni skladi. S popularizacijo svetovnega spleta pa je postalo dostopno tudi fizičnim osebam, ki z osnovnim znanjem programiranja lahko realizirajo poljubno zapleteno trgovalno strategijo, ter na njeni podlagi izvajajo posle z relativno majhnim denarnim vložkom (npr. z 100 ameriškimi dolarji).

2.2.6 Dolgoročnost

Pri ročnem trgovanju težko zagotovimo konsistentno sledenje strategiji zaradi človeškega faktorja (prisotnost čustev in časovnih omejitev). Ker se računalniški program ne utrudi, se trgovalna strategija izvaja na enak način tudi po daljšem času.

2.2.7 Optimizacija parametrov

Trgovalne strategije delujejo po določenih pogojih, t.j. ko določene izračunane vrednosti iz podatkov dosežejo nek prag za vstop v posel ali izstop iz le-tega. Te izračune in pragove lahko poljubno spreminjamo, toda ročna simulacija za vsako spremembo je praktično nemogoča. Če pa gre za algoritem, ga lahko testiramo na istih podatkih z različnimi parametri, da ugotovimo, katere vrednosti so vsaj približno optimalnejše. Problem nastane, če želimo strategijo optimizirati po večjem številu parametrov, saj velikost prostora, ki ga moramo preiskati, raste eksponentno s številom parametrov.

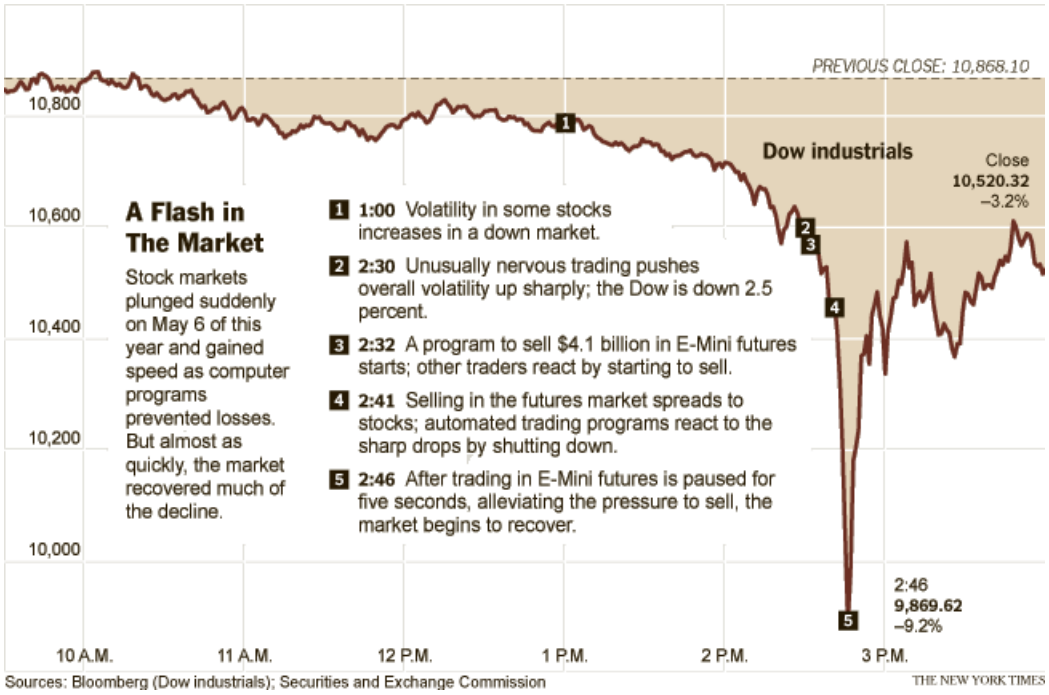
2.3 Slabosti algoritemskega trgovanja

Kljub temu, da lahko algoritem testiramo na obstoječih podatkih in da v programski kodi definiramo, kako naj se program odziva v določenih posebnih situacijah (npr. izguba povezave s strežnikom, majhna likvidnost, veliki premiki cene instrumenta), lahko med

delovanjem pride do situacij, ki niso bile predvidene. Zato je kljub dobro testirani kodi potrebno algoritem redno spremljati - ali v vseh urah delovanja ali pa v rednih intervalih. Ker pa še vseeno algoritem lahko naredi veliko poslov preden v situacijo poseže človek, so posledice lahko pogubne za sredstva na trgovalnem računu [5].

Poleg tega lahko pride do odpovedi fizičnih komponent, izgube povezave, ter drugih tehničnih težav, kjer mora posredovati človek. Prav tako algoritmi niso sposobni pravega učenja in jih je potrebno občasno prilagoditi novim situacijam. Z avtomatizacijo neke trgovalne strategije se tako delo še ne konča.

Veliko število aktivnih algoritmov na borzah pa lahko privede do domino efekta, kot je bil na primer t.i. *flash crash* 6. maja leta 2010, ko so borzni indeksi zanihali za rekordno vrednost v tako kratkem času, kot prikazuje slika 2.2.



Slika 2.2 Vrednost borznega indeksa DJIA na dan 6. maja 2010 [6].

Zaradi podobnih incidentov je bilo algoritemsko trgovanje deležno veliko kritik. Z upoštevanjem dejstva, da je večino vsega trgovanja posledica odločitev programov, je borza izgubila osnovno poslanstvo – povezovanje podjetij, ki za uresničitev ciljev potrebujejo denarna sredstva, ter investitorji, ki svoj denar želijo dati v uporabo.

2.4 Metode algoritemskega trgovanja

Ker je danes borza informatizirana, za dejansko izvedbo poslov skrbijo algoritmi, ki medsebojno povezujejo naročila – prodajalec nekega vrednostnega papirja odda naročilo za prodajo, kupec naročilo za nakup, sistem pa ju poveže.

V tem poglavju pa se ne osredotočamo na tehnični vidik izvedbe poslov, temveč na različne metode oziroma načine trgovanja, s katerimi akterji na trgu poizkušajo ustvariti dobiček. Te metode niso značilne za ročno trgovanje in nekaterih brez uporabe računalnikov praktično ne bi bilo mogoče izvajati.

2.4.1 Statistično trgovanje

Ta metoda se zanaša tako na statistične zakonitosti gibanja cene, kot tudi na druge podatke o nekem vrednostnem papirju. Dobro poznana fraza je, da kupujemo poceni in potem prodamo drago (angl. *"buy low, sell high"*). Na podlagi zgodovinskih podatkov poizkušamo ugotoviti, v katerih pogojih je primeren vstop v posel, ter kdaj ta posel zaključiti. To so predvsem podatki o gibanju cene, lahko pa vključimo tudi druge, v kolikor so le-ti dostopni. Ta metoda je najbolj popularna med končnimi uporabniki oziroma malimi investitorji, predvsem zato, ker je lahko dostopna. Veliko število podjetij nudi storitve borznega posredovanja, odprtje trgovalnega računa je enostavno, zaradi ponujenega vzvoda pa je trgovanje mogoče že z majhnih vložkom (npr. z 100 ameriškimi dolarji).

2.4.2 Arbitražno trgovanje

Pri vrednostnih papirjih, ki kotirajo na večih borzah, ves čas prihaja do nesoglasja cen. Na primer na neki borzi se zaradi povečanega povpraševanja po določeni delnici poviša njena cena, na drugi borzi pa se cena praktično ne spremeni. Ker gre za isto podjetje lahko predvidevamo, da bi morali ceni biti enaki. Zato lahko v tem trenutku na drugi borzi kupimo delnice tega podjetja, na prvi pa jih prodamo in tako ustvarimo dobiček na majhni razliki v ceni. Ker je v igri več trgovalcev oziroma institucij, ki trgujejo po istem principu, je za konkurenčnost nujno potreben majhen dostopen čas do obeh borz, poleg tega pa morajo biti stroški izvedbe poslov zelo majhni, saj se dobiček ustvari na zelo majhnih cenovnih razlikah. Zaradi teh zahtev je ta metoda primerna samo za institucije ali podjetja z veliko kapitala in ne za male vlagatelje.

2.4.3 Agregacija naročil

Institucije, ki nudijo storitev borznega posredništva za male investitorje, lahko posle izvajajo na dva načina. Prvi način je, da se vsa naročila stranke neposredno izvedejo na borzi (angl. *straight-through processing*). Gre le za premostitev dveh elektronskih sistemov, kar pa je lahko storjeno ročno, ali pa je v ozadju računalniški program.

Drugi način je posredovanje naročil med strankami, torej se posamezen posel ne posreduje na borzo. To je mogoče, kadar ima institucija veliko končnih strank. Ustvari se manjša navidezna borza - računalniški program kupce poveže s prodajalci, brez izvršitve poslov na pravi borzi. Kljub temu pa je redko mogoče dobiti enako število kupcev in prodajalcev, zato mora institucija ščititi svojo izpostavljenost. V tem primeru institucija na borzi naredi večje naročilo, katerega vrednost ustreza presežku kupcev ali prodajalcev.

Primer take agregacije naročil oziroma ščitenja izpostavljenosti lahko vidimo v podatkih prometa, ki ga institucija izvede na borzi. Borzni posrednik Dukascopy[5] nudi prost dostop do teh podatkov. Iz tabele 2.1 je razvidno, da se izpostavljenost izravna v določenih intervalih v vrednosti večkratnikov po 750 tisoč EUR.

Čas	Vrednost poslov v Evrih
15.10.2012 06:23:51.709	3,000,000
15.10.2012 06:23:52.368	4,500,000
15.10.2012 06:23:52.918	3,000,000
15.10.2012 06:23:53.458	3,000,000
15.10.2012 06:23:53.638	3,750,000
15.10.2012 06:23:54.017	3,000,000
15.10.2012 06:23:54.568	3,750,000

Tabela 2.1 Podatki izravnalnih poslov borznega posrednika Dukascopy za instrument EUR/USD, 15. oktobra 2012 [7].

2.4.4 Visokofrekvenčno trgovanje

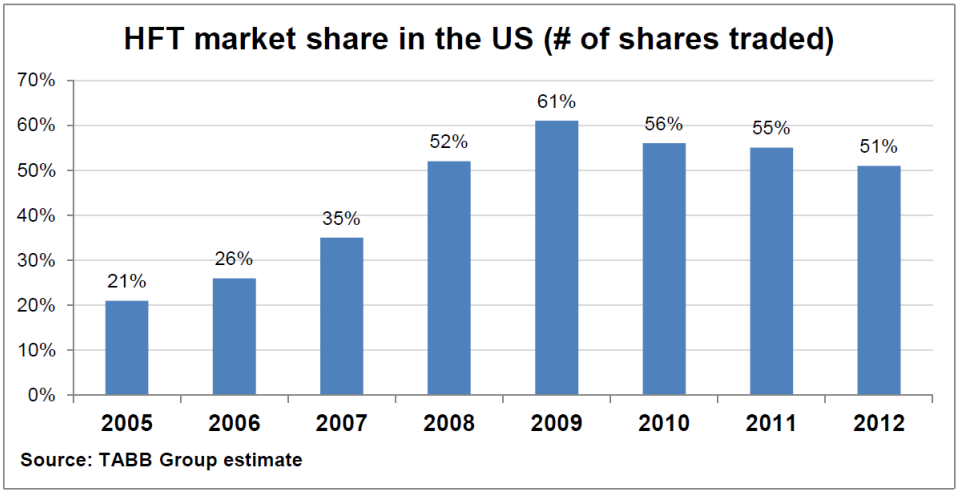
Pri visokofrenkvenčnem trgovanju [8] [9] (angl. *high-frequency trading*) gre za kombinacijo različnih metod. Poglavitnega pomena je število poslov, ki lahko doseže tudi nekaj

[5] Dukascopy Bank SA, http://www.dukascopy.com

deset tisoč na dan. Vsak posel je zaključen v zelo kratkem času - ki se lahko meri v milisekundah in ponavadi ni daljši od nekaj minut. Poleg uporabe za medborzno arbitražo se to metodo uporablja za ustvarjanje dobička na manjših spremembah cen delnice.

Primer delovanja algoritma za visokofrekvenčno trgovanje: algoritem zazna povečano povpraševanje po nekem vrednostnem papirju (na primer glede na povečanje pogostosti premikov cene ali pa spremembe likvidnosti). Algoritem zato hitro kupi ali proda večje število tega vrednostnega papirja, z namenom, da se bo ta trend nadaljeval - torej bo mogoče ta posel zaključiti z dobičkom. Pričakovanje je namreč, da bodo trend opazili tudi drugi udeleženci trga, glavni namen takega posla pa je, da se lahko ta isti vrednosni papir nekaj sekund ali minut kasneje proda tistim, ki niso bili tako hitri.

Hitrost izvajanja poslov se je v zadnjem desetletju (od 2000 do 2010) iz nekaj sekund zmanjšala na milisekunde. Ta metoda trgovanja je vir večine prometa na borzah danes, kar prikazuje graf na sliki 2.3.



Slika 2.3 Odstotek poslov, ki izvirajo iz visokofrekvenčnih trgovalnih metod po letih [10].

2.4.5 Tehnološke potrebe

Za izvajanje velikih poslov, še posebej kadar je dostop do trenutnih cen in hitra izvedba poslov kritičnega pomena, je nujno potreben direkten dostop do borze, torej brez posrednikov. Poleg tega so končni stroški trgovanja manjši, kljub temu da je mesečna naročnina za tak dostop relativno visoka (nedosegljiva oziroma neekonomična za male vlagatelje).

Pomemben je tudi hiter dostopni čas do trgovalnega strežnika - med prejetjem najnovejših informacij o instrumentu, izračunom mogočega posla ter pošiljanje tega posla v izvedbo se lahko cena že spremeni, kar pa lahko pomeni razliko med dobičkom ali izgubo. Iz tega razloga se institucije odločajo za postavitev računalniških sistemov čim bližje borzi, če je mogoče kar v sosednji stavbi [11]. Poleg tega želijo trgovalci, ki izvajajo arbitražne posle, doseči čim hitrejši dostopni čas med borzami, torej med velikimi razdaljami. Ta tekma je tudi eno izmed gonil za napeljavo novega trans-atlantskega komunikacijskega kabla, ki naj bi zagotavljal za okoli 10 milisekund hitrejšo komunikacijo [12].

2.5 Dostopne platforme za trgovanje

Za neposredno trgovanje na borzi je potreben poseben dostop, ki pa je omogočen samo institucijam. Tovrstno trgovanje urejajo posebna pravila [13]. Za izvajanje trgovalnih strategij institucije uporabljajo namenske programe, ki pa niso prosto dostopni. Te institicuje lahko trgujejo samo za lastne potrebe, ali pa delujejo kot posredniki za končne uporabnike - male investitorje. Mali investitorji dostopajo do storitev borznih posrednikov z namenskimi programi, ki omogočajo dostop do podatkov trga ter upravljanje s posli (oddajanje, pregledovanje in zapiranje). Za tak program pa ni nujno, da omogoča tudi programsko izvajanje trgovalne strategije - to omogočajo le nekatere platforme.

Opisane platforme so znane predvsem za trgovanje na valutnih trgih, toda nekateri posredniki (še posebej večji) poleg valutnega trga ponunajo trgovanje tudi z delnicami, borznimi indeksi ter dragimi kovinami in drugimi surovinami.

2.5.1 MetaTrader

MetaTrader je najbolj poznano ime v svetu algoritemskega trgovanja za male investitorje. Programski sistem je izdelalo podjetje MetaQuotes Software, ki je leta 2000 zaslovelo s programom FX Charts. Prednost tega programa je bila združljivost z operacijskim sistemom Microsoft Windows, torej za trgovanje ni bilo več potrebno imeti strojne namenske opreme. V letu 2002 so predstavili programski sistem MetaTrader, ki je poleg izvajanja ročnih poslov podpiral tudi izvajanje uporabniških algoritmov. Trgovalno logiko je mogoče napisati v programskem jeziku MetaQuotesLanguage ali MQL [14], ki je podoben programskemu jeziku C++. Koda se delno prevede (v t.i. “byte-code”), to pa potem izvaja navidezni stroj, podobno kot pri Javi. Prve različice programskega jezika so pod-

pirale samo osnovne funkcije, v verziji MQL4 pa je postal bolj uporaben. Kljub temu so odsotne možnosti programiranja po modernih principih, na primer ne podpira razredov ali kazalcev. Po drugi plati že podpira vključevanje dinamičnih knjižnic, torej lahko algoritem realiziramo v nekem drugem programskem jeziku, ter v jeziku MQL poskrbimo samo za komunikacijo med MetaTrader sistemom in dinamično knjižnjico. V letu 2010 pa so izdali tudi peto različico tega sistema, ki med drugimi izboljšavami vključuje tudi objektno programiranje. Slaba lastnost te nadgradnje je, da z novo različico niso združljivi algoritmi, napisani v jeziku MQL4.

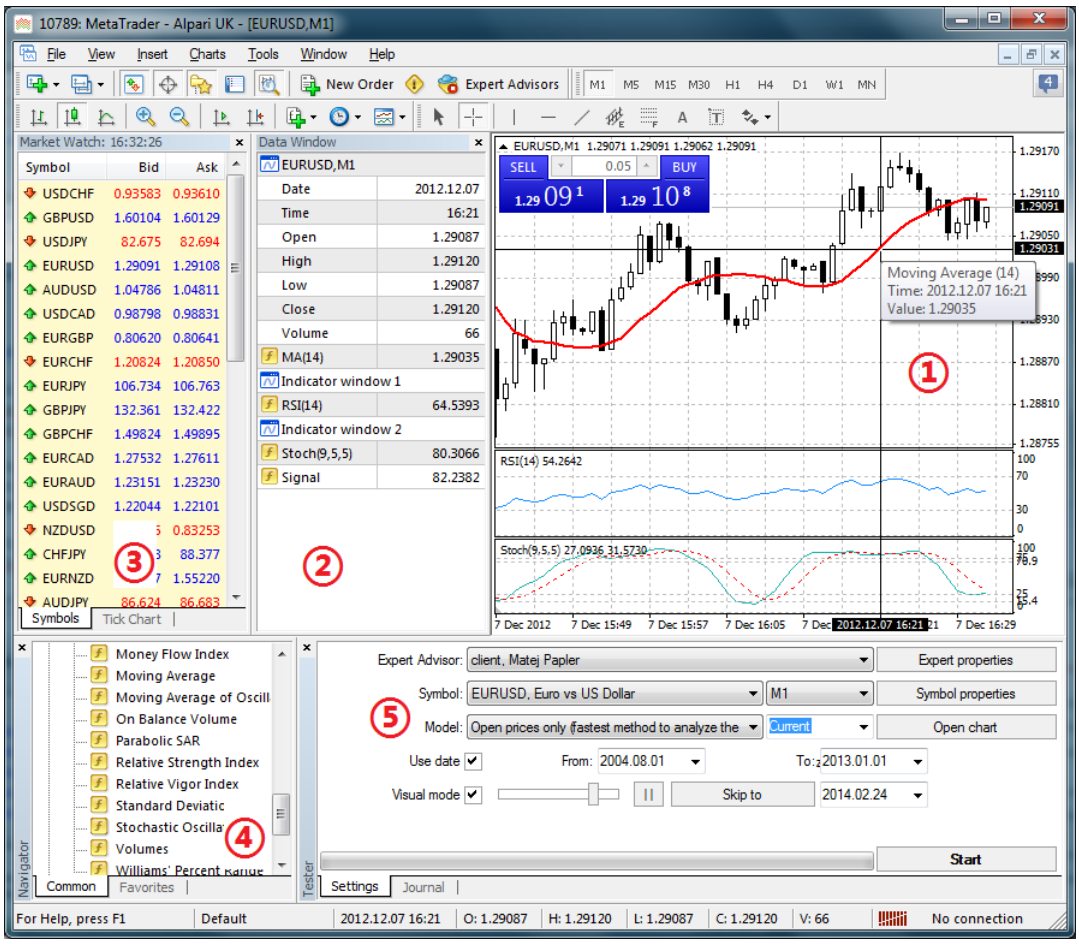
Sistem je plačljiv, toda brez omejitev dostopen vsem institucijam, ki to funkcionalnost želijo ponuditi svojim strankam. Uporaba programa je brezplačna za končnega uporabnika (licenčnina se odraža v trgovalnih stroških). Iz tega razloga je med končnimi uporabniki najbolj razširjen program tako za ročno, kot tudi algoritemsko nadzorovano trgovanje.

Na sliki 2.4 je prikazano glavno okno, pomembnejši gradniki pa so oštevilčeni:

1. prikaz grafa trenutno izbranega instrumenta oziroma vrednostnega papirja,
2. podatkovno okno, ki prikazuje natančne številčne vrednosti,
3. seznam instrumentov, ki so na voljo pri tem posredniku,
4. seznam vgrajenih indikatorjev ter po meri narejenih (lastnih) indikatorjev, skript in ExpertAdvisor-jev,
5. orodje za simulacijo trgovalne logike (ExpertAdvisor-jev).

2.5.2 Modulus

Program Modulus M4 podjetja Modulus Financial Engineering, Inc. je novejši na trgu, v času pisanja tega besedila redko uporabljen s strani borznih posrednikov. Je pravzaprav obsežna zbirka različnih modulov, napisanih v programskem jeziku C#. To so moduli za pridobivanje podatkov o trenutnih cenah za izvrševanje poslov na različnih sistemih, zbirke knjižnic za analizo podatkov, pridobivanje novic v obliki RSS iz različnih virov, itd. Na voljo je tako brezplačna različica za osebno uporabo, kot tudi dve različni plačljivi različici: cenejša za razliko od brezplačne dovoljuje uporabo v komercialne namene, dražja pa vključuje tudi celotno izvorno kodo tega programskega sistema.



Slika 2.4 Uporabniški vmesnik programa MetaTrader4.

2.5.3 VT Trader

VT Trader je razvilo podjetje Visual Trading Systems, ter je bil v začetku popularna alternativa sistemu MetaTrader. Je nekoliko naprednejši, vendar z algoritmskega stališča ne podpira ničesar takšnega, česar ni mogoče realizirati tudi v MetaTrader-ju. Omogoča izdelavo lastnih indikatorjev ter trgovalnih strategij v posebnem opisnem jeziku.

2.5.4 NinjaTrader

Od ostalih rešitev se razlikuje v tem, da je bil prvi, ki je omogočal programiranje v programskem jeziku C#. To je bila velika prednost, saj se za realizacijo trgovalnega algoritma ni potrebno uvajati v poseben programski jezik, ter je enostavno vključiti razne funkcije, ki so že napisane v tem jeziku. Programski sistem so brezplačno ponudili

nekateri večji posredniki in pridobiva na popularnosti.

2.5.5 API dostop

Nekateri borzni posredniki ponujajo t.i. API dostop (angl. *application programming interface*), preko katerega je mogoče pridobivati trenutne podatke o trgu ter izvajati posle. S tem lahko uporabnik na poljuben način izdela aplikacijo za izvajanje trgovalne strategije in ga preko API vmesnika poveže s trgovalnim strežnikom.

2.5.6 Sledenje sistemom

V zadnjih letih se uveljavlja t.i. sledenje sistemom (angl. *trade following* ali *trade mirroring*). Ponudnik izvaja razne trgovalne strategije in posle prikaže naročnikom. Za te posle ni nujno, da so bili dejansko izvedeni, temveč so samo ideje za posle, katere lahko trgovalci izvajajo s svojimi sredstvi. Kadar gre za več strategij, si lahko uporabnik izbere seznam tistih, ki jih želi uporabljati. Za vsako pa določi parametre izpostavljenosti, s čimer posle prilagodi velikosti svojega računa. Te posli se izvajajo avtomatično, torej brez spremljave naročnika. Tovrstne algoritemsko generirane posle pogosto brezplačno nudijo kar borzni posredniki - njihov cilj je, da uporabniki čim več trgujejo, saj od vsakega posla posrednik dobi provizijo.

2.6 Programski Jezik MQL

V programu MT4 je za programiranje dodatnih funkcionalnosti na voljo programski jezik MQL4 [14]. Jezik je proceduralen in je sintaktično podoben programskemu jeziku C, vendar omejen na precej bolj osnovne podatkovne tipe. Osnovne podatkovne strukture so:

- `int` (celo število s predznakom, 32-bitno),
- `double` (celo število s predznakom, 64-bitno),
- `string` (kodiranje ASCII, brez podpore unicode).

S podatkovnim tipom `int` sta povezana še `datetime` (zapisan kot t.i. “unix time”, ki predstavlja število sekund od 1.1.1970) in `color` (trije od štirih 8-bitnih delov določajo RGB barvo 24-bitne globine). Prednost teh dveh je le v drugačni predstavitvi uporabniku, na primer da se namesto vnosa števila prikaže izbor datuma iz koledarja oziroma

izbor barve iz palete. Obstajajo tudi tabele, ki lahko vsebujejo podatke kateregakoli tipa in imajo do tri dimenzije.

Programski jezik ne podpira definicije podatkovnih struktur (`struct`), objektov ali kazalcev, zaradi česar je razvoj bolj kompleksnih sistemov nekoliko otežen.

Vsak MQL4 program ima lahko definirane vhodne parametre - ko program zaženemo, se prikaže okno v katerega jih vpišemo - kot glavni vir podatkov pa je branje tečaja iz grafa instrumenta ter branje drugih indikatorjev. Omogočeno je tudi delo z datotekami (branje in pisanje binarnih ali tekstovnih datotek), ter vključevanje programskih knjižnic DLL.

Obstajajo štiri vrste datotek s kodo:

- Indicator: Na osnovnem ali pomožnem grafu lahko izrisujejo krivulje, kot tudi objekte (črte, like, tekst). Ne morejo odpirati ali zapirati poslov. Vsak graf instrumenta ima lahko več indikatorjev.

- Expert Advisor (EA): Namenjeni so za avtomatsko trgovanje. Na vsak graf instrumenta lahko postavimo samo enega. Poleg zmožnosti odpiranja in zapiranja poslov lahko tudi izrisuje objekte na grafu. Ko ga zaženemo, se izvaja, dokler ga ročno ne izključimo.

- Script: Zelo podoben Expert Advisorju, z razliko da se ne izvaja do uporabnikovega preklica, ampak se koda požene samo enkrat in se potem zaključi.

- Library: Vsebuje definicije spremenljivk, konstant in funkcij, da lahko iste dele kode uporabimo v različnih programih.

Vsaka vrsta programa (indicator, EA ali script) ima definirane tri osnovne funkcije, ki omogočajo delovanje. To so funkcije:

- `init()` se kliče ob inicializaciji programa (dvojni klik z miško na ikono datoteke, oziroma če ikono povlečemo in spustimo na graf). Tukaj na primer inicializiramo spremenljivke, preberemo konfiguracijske podatke iz datoteke, oziroma programih tipa Indicator definiramo katere črte se bodo izrisovale na grafu (ter jim določimo barvo, debelino in slog).

- `deinit()` se kliče ob odstranitvi programa iz grafa, oziroma v primeru programov tipa script, ko se funkcija start zaključi.

- `start()` v programih tipa Indicator ali EA se izvede takoj po klicanju funkcije `init`, ter potem vsakič ko se spremeni vrednost tečaja - ko MetaTrader prejme nov podatek o trenutni ceni instrumenta. Za vrsto datoteke Script se izvede samo enkrat.

3 Nadgradnja programskega paketa MT4

Poleg predstavitve algoritemsko nadzorovanega trgovanja želimo tehnični vidik algoritemsko nadzorovanega trgovanja prikazati tudi na praktičen način, tako da izboljšamo enega od dostopnih programskih paketov opisanih v poglavju 2.5. Izbrali smo si programski paket MT4, ker je trenutno najbolj razširjen. Identificirali smo nekatere pomanjkljivosti in izdelali programsko rešitev.

3.1 Razlogi za nadgradnjo

Programski paket MT4 ima nekaj pomanjkljivosti, ki jih želimo odpraviti. Te so sledeče:

1. Za izvajanje vsake strategije moramo odpreti nov graf s ceno, ki zaseda prostor v delovnem pomnilniku in procesorski čas. Želimo torej omogočiti, da na nekem grafu poganjamo več strategij hkrati.

2. Ker je izvajanje strategij lahko zahteven proces, želimo ločiti del logike in dejanskega izvajanja poslov. S tem lahko omogočimo zrcaljenje poslov na več trgovalnih računih.

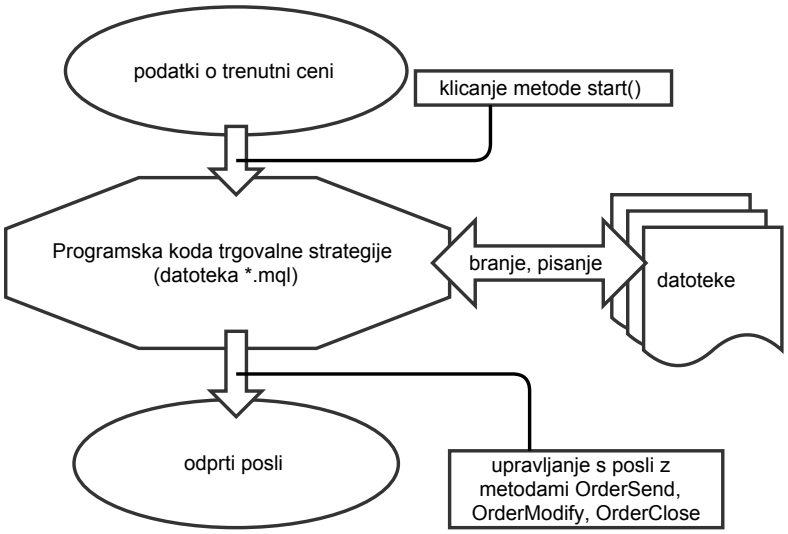
3. Kadar smo finančno izpostavljeni na trgu, ima lahko odpoved strojne opreme

ali povezave s strežnikom hude posledice za imetnika trgovalnega računa. Omogočiti moramo, da lahko neka strategija teče na več računalnikih hkrati - aktivno izvajanje na primarnem računalniku ter samodejen preklop na sekundarnega v primeru izpada prvega.

4. Pri simulaciji strategije ni možnosti, da bi simulacijo izvajali na več grafih hkrati. Kadar te strategije trgujejo neodvisno, lahko naredimo enostaven izpis denarnega stanja po intervalih, izvedemo strategije eno za drugo, ter potem datoteke s temi podatki združimo, da lahko ocenimo dobičkonosnost in tveganje. Če pa se strategije povezujejo, na primer da je odpiranje novega posla odvisno od trenutne izpostavljenosti drugih strategij ali pa vrednosti njihovih notranjih spremenljivk, moramo omogočiti, da strategije medsebojno komunicirajo ter se simulirajo sočasno.

3.2 Izbrane rešitve

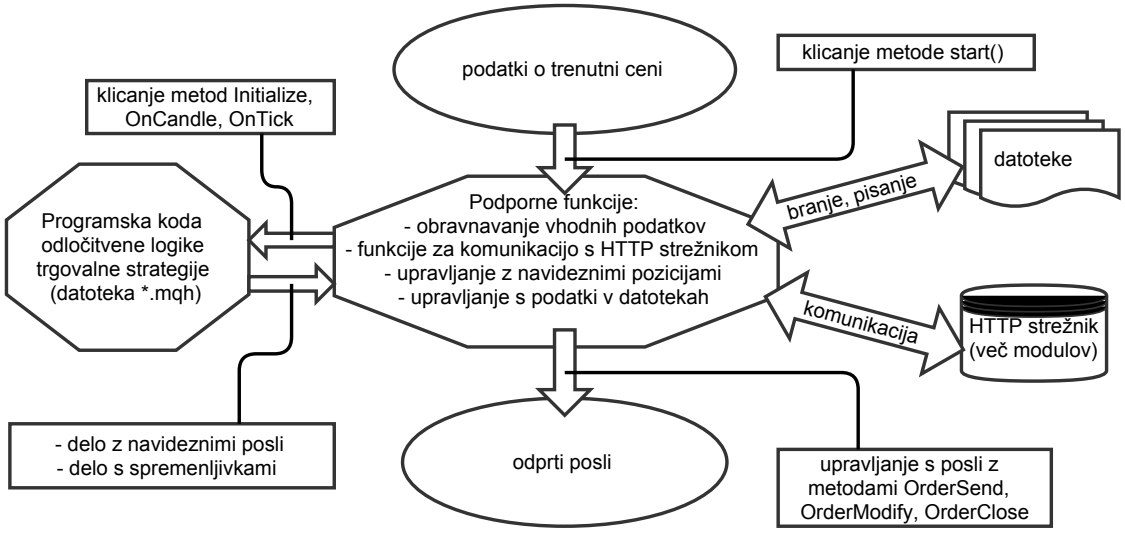
Pri izdelavi EA-ja mora programer paziti na morebitne napake, ki se lahko zgodijo ob nalaganju ali shranjevanju podatkov, ter delom s posli. Na sliki 3.1 je prikazana shema običajnega Expert Advisorja.



Slika 3.1 Vizualna predstavitev izdelave logike trgovalne strategije v programskem jeziku MQL4.

Odločili smo se, da bomo logični del trgovalne strategije, torej algoritme za odločanje o odpiranju in zapiranju poslov, ločili od sistemskega dela – branja in pisanja podatkov, preverjanja napak ter beleženja okoljskih spremenljivk. Tako se ob programiranju strategije posvečamo samo dejanski odločitveni logiki.

Glede na strukturo programskega jezika MQL4 smo se odločili za izdelavo več knjižnic, ki bodo služile delu na sistemskem nivoju. Tako olajšamo izdelavo trgovalne logike, saj je ločena od sistemske funkcionalosti, kot prikazuje slika 3.2.



Slika 3.2 Vizualna predstavitev ločitve trgovalne logike od podpornih funkcij.

Definirali smo, da se trgovalna logika izvaja s pomočjo štirih funkcij:

- `Initialize()` - kliče se znotraj funkcije `init()`, ko so naloženi vsi parametri ter podatki stanja strategije,

- `OnCandle()` - kliče se znotraj funkcije `start()`, kadar se je na grafu pojavil nov časovni interval oziroma svečka[1],

- `OnTick()` - kliče se znotraj funkcije `start()`, ob vsaki spremembi cene,

- `Deinitialize()` - kliče se znotraj funkcije `deinit()`, torej če želimo v trgovalni logiki izvesti nekaj ob končani simulaciji oziroma kadar strategijo onemogočimo,

- `ApplyParam( string name, double value )` - kliče se znotraj funkcije init() in poskrbi za to, da se shrani oziroma naloži vrednost nekega parametra.

[1] Za prikaz grafa s ceno se mnogokrat uporabljajo t.i. japonske svečke (angl. *japanese candlesticks*), kjer vsaka svečka prikazuje ceno v nekem časovnem obdobju, na primer minuto, 5 minut, 15 minut, uro, dan, itd.

Trgovalna logika je zapisana v datoteki s končnico .mqh (na primer `logika.mqh`), ki mora imeti definirane te metode. Za dejansko uporabo tako definirane logike moramo narediti še dva MQL programa - tipa Indicator in ExpertAdvisor. Za Indicator ustvarimo datoteko (na primer `indicators/logika.mq4`), v kateri z ukazom `include` vključimo datoteko s podpornimi funkcijami za indikator `base.indicator.mqh` ter datoteko z logiko (na primer `logika.mqh`). Podobno storimo za ExpertAdvisor, le da tu vključimo datoteko s podpornimi funkcijami `base.ea.mqh`.

V tako ločeni trgovalni logiki ni dovoljeno uporabljati obstoječih funkcij za delo s posli, temveč moramo uporabljati funkcije iz naše nadgradnje. Nekatere od teh funkcij so za uporavljanje s t.i. navideznimi pozicijami, ki so le zapisi v tabeli. Nato pa v nastavitvah posameznega sistema določimo kako in na katerih trgovalnih računih se te navidezne pozicije izvedejo kot pravi posli. Trgovalni logiki so na voljo tudi podporne funkcije za delo s t.i. obstojnimi spremenljivkami. Vsaka številska vrednost, ki se med izvajanjem algoritma spreminja in vpliva na delovanje, se mora shraniti na tak način, da bo algoritem pravilno deloval tudi ob ponovnem zagonu sistema ali pa ob prenosu izvajanja na drug računalnik v primeru odpovedi. Tako se programerju trgovalne logike ni potrebno ukvarjati s tem, ali bo neka spremenljivka shranjena v lokalno datoteko ali na HTTP strežnik, ter na katerih trgovalnih računih bo nek posel dejansko odprt. Podporne funkcije, ki so direktno namenjene uporabi v trgovalni logiki, lahko razdelimo v naslednje sklope:

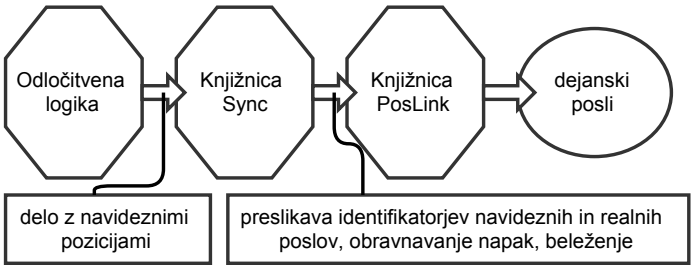
- Priklic vrednosti vhodnih parametrov programa, ki so bili lahko podani na dva načina: kot dejanski parameter programa EA oziroma indikatorja, ali pa kot parameter v nizu znakov, ki je bil posredovan iz neke konfiguracijske datoteke.

- Delo z navideznimi pozicijami, torej ustvarjanje novih pozicij ter zapiranje obstoječih. Predvidevamo da bo potrebno navidezne pozicije združevati v skupine.

- Branje in pisanje obstojnih spremenljivk, ki služijo shranjevanju notranjega stanja trgovalne logike (množica podatkov, ki vpliva na odločanje v prihodnosti).

Skupek podatkov, na podlagi katerih deluje neka strategija, torej seznam navideznih pozicij in obstojnih spremenljivk, poimenujemo “stanje strategije”. To stanje podporne funkcije zapišejo v lokalno datoteko oziroma shranijo na HTTP strežnik, kjer je dostopna redundančnim HTTP strežnikom in posledično redundančnim terminalom, ki poganjajo

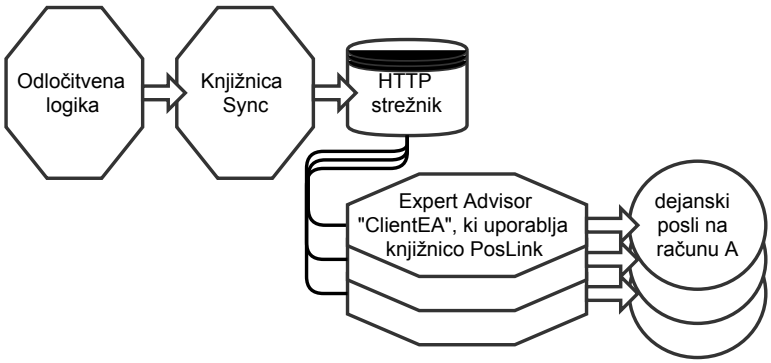
isti algoritem. Navidezne pozicije je mogoče realizirati v dejanske posle direktno, ali pa jih posredno izvajamo na drugih terminalih in s tem omogočimo funkcionalnost zrcaljenja poslov.



Slika 3.3 Neposredno odpiranje navideznih pozicij.

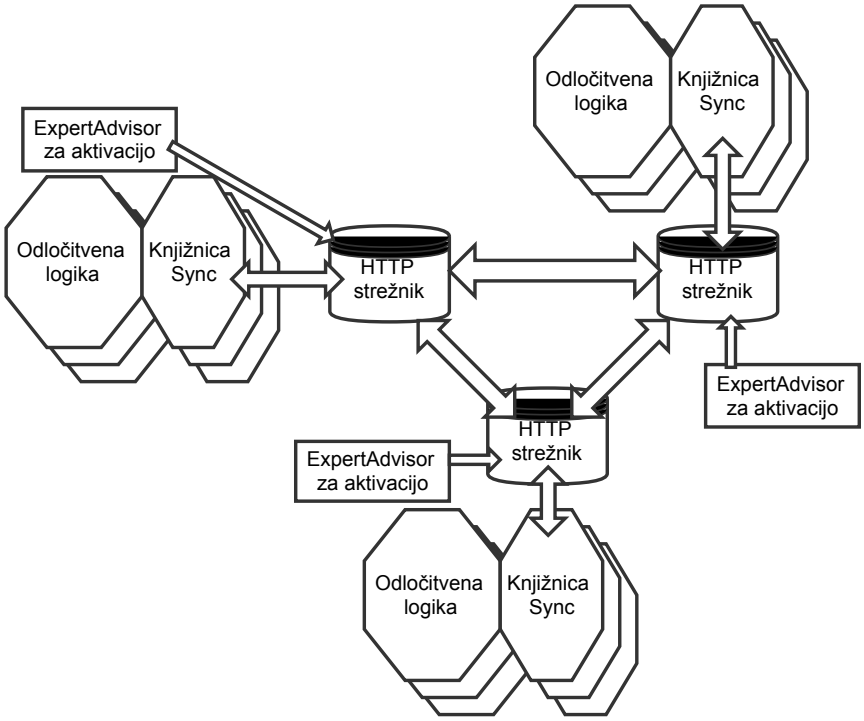
Na sliki 3.3 je prikazana shema neposrednega odpiranja poslov - primer, da v Expert Advisorju te strategije (na primer `Logika.mq4`) v knjižnici PosLink vključimo tudi direktno odpiranje poslov.

Direktno odpiranje pa ni cilj našega sistema. Želimo omogočiti, da na enem grafu hkrati teče več trgovalnih strategij in da vsaka svoj seznam navideznih pozicij zapisuje na neko mesto. Te sezname pa lahko nato z uporabo namenskega EA-ja pretvorimo v dejanske posle na enem ali več trgovalnih računov hkrati. Iz tega razloga seznam navideznih pozicij shranjujemo na HTTP strežnik, kot je prikazano na sliki 3.4.



Slika 3.4 Posredno odpiranje navideznih pozicij na več trgovalnih računih hkrati.

Poleg posrednega odpiranja pa moramo misliti tudi na redundančnost, torej da se



Slika 3.5 Povezovanje več skupin.

isti algoritem z istimi parametri izvaja na drugem računalniku, kar je prikazano na sliki 3.5. Vsak algoritem deluje neodvisno od drugih, ter periodično posreduje stanje lokalnemu HTTP strežniku – se sinhronizira. Ob vsaki sinhronizaciji po poizkusu zapisovanja trenutnega stanja tudi prebere stanje iz strežnika. Obstaja namreč možnost, da je nek primerek istega algoritma na računalniku z večjo prioriteto prišel do drugačnega stanja, torej se trenutno stanje z manjšo prioriteto zavrže.

Ugotavljanje trenutno aktivnega računalnika omogoča poseben Expert Advisor, ki teče na vseh računalnikih na nekem trgovalnem računu. Komunikacija med terminali poteka preko parametrov čakajočih naročil. Ugotovili smo, da je to najbolj zanesljiv način, saj dva računalnika, ki sta povezana v internet preko različnih ponudnikov, lahko medsebojno izgubita povezavo, kljub temu da imata oba povezavo s trgovalnim strežnikom. Za namen komunikacije vsak terminal ustvari lastno čakajoče naročilo s takšnimi parametri, da se nikoli ne bi izvršilo – zaradi izjemne razlike med naročeno ceno in trenutno ceno. Nato pa v vrednost cene za omejevanje izgube in vnovčevanja dobička (angl. *stop-loss*

& take-profit) vpisuje trenuten čas kot "unix time" ter status aktivnosti kot število 0 ali 1. Vsak terminal ima določeno številko prioritete - primarni ima prioriteto T=1, sekundarni prioriteto T=2, ter poljubno mnogo redundantnih računalnikov, ki bi bili označeni z večjimi številkami, torej nižjimi prioritetami. Vsi terminali vsako sekundo preko teh čakajočih naročil preverjajo prisotnost drugih terminalov. Če terminal ugotovi, da noben terminal z višjo prioriteto od njega v določenem času ni osvežil svoje aktivnosti, se aktivira, hkrati pa svoj status aktivnosti postavi na 1, drugim terminalom pa na 0.

Na primer: Terminal T=1 ima svoje čakajoče naročilo na instrumentu EUR-USD pri tečaju 10000, take-profit pri 13921,33501 (vrednost "unix time" za 11. februar 2013 ob 15:45:01), stop-loss pa 1,0000 s čimer ostalim terminalom sporoča, da je aktiven. Svoj čas aktivnosti periodično posodablja. Če pa bi ob času 15:48:00 bila ta vrednost še vedno ista, bi terminal T=2 to razumel kot da je terminal T=1 izgubil povezavo, ter bi prevzel trgovanje. Ko aktiven terminal ugotovi, da je prisoten drug terminal z večjo prioriteto, mu preda aktivnost.

3.3 Vodenje razvoja

Kot pomoč pri razvoju smo uporabili sistem za vodenje razvoja Subversion[2], skupaj z vmesnikom TortoiseSVN. Zaporedna številka revizije je z notacijo `$Revision$` [15] uporabljena v datoteki izvorne kode `svn.mqh`, klic metode `SvnRevision()` pa vrne vrednost trenutne verzije kode, ki se lahko uporablja za prikaz in beleženje v dnevniške datoteke.

3.4 Serializacija v tekst

Za dosego ciljev mora sistem omogočati izvoz in uvoz trenutnega stanja, t.j. vrednosti internih spremenljivk, seznam pozicij in seznam skupin pozicij. Za enostavnejšo uporabo in morebitno ročno urejanje smo izbrali tekstovno obliko izvoza. Jezik MQL ima že definirane funkcije za delo z CSV datotekami, toda ta format ni optimalen za shranjevanje tako raznolikih podatkov, kot jih pričakujemo za različne trgovalne strategije. Kot format izvoza bi lahko uporabili katerega od popularnih formatov XML ali JSON, toda ta dva v osnovi zahtevata drevesno strukturo, katero pa je v jeziku MQL težko realizirati, saj ne podpira objektov ali kazalcev. Odločili smo se za prilagojen format, za katerega je enostavno narediti algoritem za pisanje in branje, ter je hkrati enostavno berljiv človeku.

[2] Apache Subversion je odprtokodni programski paket. Pogosto se kot krajše ime uporablja "SVN".

Primer izvoza:

```
ikepaIndicatorState
Variables
    StateVersion = 51
    LastCandle = 2014.02.12 08:26
    CandleCount = 73
    Symbol = EURUSD
    Timeframe = M1
Families
    LastID=2 Count=2
    1~1 L*0 re=999999
    2~2 S*0 re=999999
Positions
    LastID=32 Snake=17,12,17,15
    30~2 S @o2014.02.12_08:17~1.36650
    31~2 S @o2014.02.12_08:18~1.36648
    32~2 S @o2014.02.12_08:19~1.36648
PosLink
    PosLink Version=0 Count=3
    123~30 #0 S*1 @o2014.02.12_08:17~0
    123~31 #0 S*1 @o2014.02.12_08:18~0
    123~32 #0 S*1 @o2014.02.12_08:19~0
History
    1392188400
    1392188520,1
    1392188580,2
    1392188760,3
    1392189000,4
```

3.5 HTTP strežnik za sinhronizacijo

MT4 podpira vključevanje dinamičnih knjižnic (DLL). Uporabili bi lahko povezovanje direktno preko protokola TCP/IP, vendar smo se odločili za uporabo višjeslojnega protokola HTTP, ker lahko do strežnika dostopamo s spletnim brskalnikom in je izdelava vmesnika veliko bolj enostavna, kot da bi izdelali ločen program za povezovanje s strežnikom in upravljanje s podatki.

V programskem jeziku C# smo izdelali enostaven HTTP strežnik[3]. Zasnovan je

[3]Prva verzija z osnovno funkcionalnostjo je bila sprva narejena v programskem jeziku PHP na strežniku Apache 2.1, toda delovanje je bilo zelo počasno.

večnitno, da lahko hkrati postreže več zahtevkov. Poleg strežbe datotek omogoča izvajanje modulov, ki vhodne podatke dobijo preko URL-ja ali HTTP `POST` ukaza.

Moduli so definirani kot datoteke s C# izvorno kodo v podmapi `scripts`. Da ustvarimo modul, definiramo nov razred. Ime tega razreda je uporabljeno kot del naslova URL, na primer, razred `Abc` bo dostopen na naslovu `http://localhost/scripts/Abc`. Razred mora dedovati po razredu BaseScript, poleg tega pa moramo definirati naslednje metode:

- `Handle(Request r)`: Metoda se kliče ob HTTP zahtevku strežnika. Tukaj poskrbimo za obravnavo vhodnih podatkov (morebitno shranjevanje) ter izpis odgovora na zahtevek.

- `Serialize(GenericWriter writer)`: Metoda v binarni zapisovalec shrani podatke, ki so relevantni za ta modul. Kliče se periodično ali pa na uporabnikovo zahtevo.

- `Deserialize(GenericReader reader)`: Metoda prebere podatke iz binarnega bralnika. Kliče se ob zagonu strežnika.

Objekt tipa `Request` vsebuje podatke o HTTP zahtevku, kot so prebrane vrednosti parametrov iz URL naslova (angl. *query string*), prebrane vrednosti iz HTTP POST ukaza (angl. *POST data*) ter naslov oddaljenega računalnika. Hkrati vsebuje metode za odgovor na HTTP zahtevo.

3.5.1 Format za izmenjavo podatkov

Pri izmenjavi podatkov med programom MQL in HTTP strežnikom lahko pride do napake in podatki niso pravilno prenešeni. Predpostavili smo, da bo sistem za detekcijo in popravljanje napak na TCP/IP sloju preprečil morebitne napake [16] v vsebini, še vseeno pa moramo poskrbeti za zaznavo prekinjenega prenosa - torej, da smo dobili le del podatkov.

To smo rešili tako, da smo podatke obdali s posebnim nizom znakov, hkrati pa definirali enostaven format za podajanje poimenskih spremenljivk (glej poglavje 3.8, asociativna tabela).

```
<DIV ID="PARSABLE">OK
_Lines: 3
VarPi: 3.14
VarName: Name Surname

Vec vrstic besedila, kot
na primer stanje strategije
ali pa preslikovalna tabela PosLink
</DIV>
```

HTML element DIV je uporabljen zato, da lahko na začetku izpišemo tudi enostavno definicijo sloga v CSS notaciji za lepši prikaz, kadar ta izpis gledamo ročno s spletnim brskalnikom.

```
<style>#PARSABLE{
border: 3px gray inset; font-family: Consolas;
white-space: pre; font-size:12px; }
</style>
```

Za delo s tem formatom (izvoz podatkov v ta format, ter uvoz podatkov ki so zapisani v tem formatu) smo v jeziku MQL definirali funkcije:

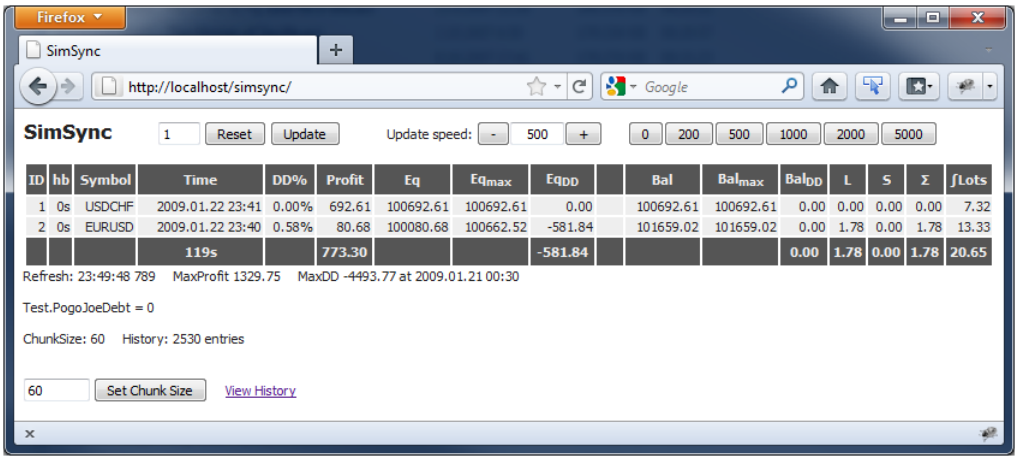
- `ParsableRequest`,
- `ParsableRequestPost`,
- `ParsableRequestHandle`.

Na strani HTTP strežnika pa smo definirali razred `Parsable`, ki vsebuje podobne funkcije.

3.5.2 Modul SimSync

To je glavni modul za sinhronizacijo simulacij. Do njega dostopamo na naslovu `http://localhost/scripts/SimSync`, uporabniški vmesnik pa je prikazan na sliki 3.6. Deluje kot semafor za posamezne terminale - skrbi, da se vse simulacije izvajajo sočasno.

Vsak EA, ki je narejen za sinhronizirano simuliranje, mora v metodi `start()` vsebovati klic funkcije `SimSyncOnTick`, ki je podrobneje opisana v poglavju 3.7.



Slika 3.6 Uporabniški vmesnik za nadzor sinhrone simulacije.

Pred zagonom sinhronizirane simulacije z ukazom `SimSync?action=reset` ponastavimo stanje simulacije.

Do modula lahko dostopa tudi uporabnik (izvajalec testa), da lahko nadzira potek kot tudi uspešnost strategije, kot je prikazano na sliki 3.6.

3.5.3 Modul SharedVariables

Za izmenjavo vrednosti med Expert Advisorji v eni instanci MT4 se uporabljajo t.i. globalne spremenljivke (angl. *global variables*). Nabor funkcij za delo s temi spremenljivkami omogoča pridobivanje in spreminjanje vrednosti. Za izmenjavo vrednosti med trgovalnimi algoritmi, ki se izvajajo na različnih instancah MT4, smo realizirali modul na HTTP strežniku, ki omogoča enakovredno funkcionalnost. Podprte so samo spremenljivke tipa `double`. Obstajajo trije ukazi za delo s spremenljivkami:

- `get` - ukaz, ki vrne vrednost spremenljivke. Izvedemo ga s klicem URLja `http://localhost/scripts/SharedVariables?action=get&k=Ime-Spremenljivke`, kot rezultat pa v obliki Parsable izpiše vrednost te spremenljivke.

- `set` - ukaz, ki postavi vrednost spremenljivke. Izvedemo ga s klicem URLja `http://localhost/scripts/SharedVariables?action=set&k=Ime-Spremenljivke&v=Vrednost`. Novo vrednost izpiše v formatu Parsable.

- `inc` - omogoča atomarno povečevanje ali zmanjševanje spremenljivke za neko vrednost. Ukaz izvedemo s klicem URLja `http://localhost/scripts/SharedVariabes?`

`action=inc&k=Ime-Spremenljivke&by=Vrednost`, ki izpiše novo vrednost v formatu Parsable. Pri ukazu `increment` lahko navedemo tudi, da je vrednost izražena v eni valuti in naj se vrednost spremenljivke spremeni glede na relativno vrednost druge valute v določenem času.

3.5.4 Modul DailyQuotes

Navedeni modul hrani podatke o dnevnih cenah instrumentov, omogoča pa pretvarjanje cen med instrumenti. Kadar želimo pretvoriti neko število enega vrednostnega papirja v drugega, izvedemo ustrezno funkcijo, pri kateri je obvezen parameter za pretvorbo tudi datum oziroma čas pretvorbe. Povezave med valutami in drugimi vrednostnimi papirji ima definirane kot usmerjen graf, po katerem z Djikstrovim algoritmom najde najkrajšo pot med dvema valutama. Torej ni potrebno, da ima podatke za vse instrumente. Za pretvorbo je pogoj le, da je graf povezan (sicer kot rezultat vrne število 0). Na primer: želimo izvedeti koliko je vreden en Evro (EUR) v Kanadskih dolarjih (CAD) na dan 4.11.2004 ob 13:58 - najbližji dnevni zapis je v tem primeru na dan 5.11.2004 ob 0:00, pretvorbo pa lahko izvede tudi, če ne obstaja direktni podatek EUR-CAD ali CAD-EUR, ampak obstaja podatek za EUR-USD in USD-CAD.

3.5.5 Modul PhoneAlert

Omogoča pošiljanje kratkega tekstovnega sporočila na mobilno napravo (SMS) z uporabo storitve SMSApi[4]. Funkcionalnost obsega konfiguracijsko stran, dosegljivo na naslovu `http://localhost/scripts/SMS?action=menu`, ter metodo `Enqueue`, s katero lahko v programski kodi enostavno pošljemo SMS.

3.5.6 Varnost izmenjave podatkov

Da smo omogočili HTTP povezovanje v programskem jeziku MQL, smo uporabili knjižnico `http51.dll`, ki pa ne podpira HTTPS povezovanja. Za šifrirano komunikacijo med računalniki preko interneta, ki zagotavljajo redundančnost, lahko uporabimo šifriran IP tunel.

[4] Dosegljivo na strani http://www.smsapi.si, storitev ponuja podjetje Ms3 d.o.o.

3.5.7 Sinhronizacija simulacije

Sama sinhronizacija deluje tako, da modul `SimSync` pozna trenuten simulacijski čas vseh primerkov programa MT4 (terminalov) oziroma ExpertAdvisorjev, ki jih hkrati izvajamo. Med izvajanjem simulacije posamezni terminali povprašujejo za dovoljenje nadaljevanja izvajanja. Ta modul skrbi, da hitrejše simulacije počakajo toliko časa, dokler jih počasnejše ne dohitijo.

Preden nek terminal izvede simulacijo strategije za neko kratko časovno obdobje, na primer nekaj minut, se poveže s strežnikom. Modul `SimSync` na podlagi simulacijskih časov drugih terminalov ugotovi, ali naj povprašujočemu terminalu dovoli nadaljnje izvajanje. Če so druge simulacije pri enakem datumu in času, strežnik odgovori pritrdilno - odgovori z nekim datumom in časom, do katerega je dovoljeno nadaljnje izvajanje, nato pa ta terminal nadaljuje simulacijo. Če pa obstaja en ali več terminalov, ki še zaostaja za drugimi, potem hitrejši terminal počaka nekaj milisekund in ponovno povpraša strežnik za dovoljenje nadaljnjega izvajanja. Torej izvajanje neke trgovalne strategije čaka, dokler je vsi ostali terminali ne dohitijo. Skupaj s poizvedbo o času srečanja, se pošljejo razne relevantne informacije, kot na primer trenutno denarno stanje in izpostavljenost.

Simulacija se izvaja v kosih (angl. *chunks*). Velikost kosa je mogoče spremeniti med izvajanjem preko uporabniškega vmesnika. Torej ni nujno, da se vsaka sekunda simulacije na vseh strategijah izvede sočasno, ampak se lahko terminali srečajo tudi na daljših časovnih obdobjih, na primer na vsakih nekaj minut, ur ali celo dni. S povečanjem časa srečanja se izvajanje simulacije pohitri, kakovost rezultatov pa poslabša.

3.6 Knjižnica PosLink

Navidezne pozicije je potrebno sinhronizirati z dejanskimi posli, za kar obstaja knjižnica `PosLink` (ime odraža povezavo med navideznimi pozicijami in realnimi posli, angl. *position linking*). Ta na podlagi preslikovalne tabele (preslikava med zaporedno številko navidezne pozicije in identifikacijsko številko posla) odpira in zapira posle. S tem načinom se lahko pozicije oziroma posle zrcali med računi – na enemu terminalu MT4 se izvaja logika strategije, ki kot rezultat izračunov v datoteko ali na HTTP strežnik piše seznam navideznih pozicij. Na drugih terminalih se izvaja ExpertAdvisor `ClientEA`, ki prebere ta seznam ter preko svoje preslikovalne tabele primerno odpira ali zapira posle, da je izpostavljenost na trgu taka kot veli strategija. Tako je mogoče na enem računu izvajati

poljubno mnogo strategij. To zmanjša uporabo pomnilnika ter obremenitev procesorja, saj na primer namesto, da bi 10 terminalov vsak poganjalo 10 strategij, te terminali samo berejo tabelo navideznih pozicij in periodično sinhronizirajo posle.

3.6.1 Serializacija v tekst

Podatki preslikovalne tabele `PosLink` se shranjujejo podobno, kot je opisano v poglavju 3.4. Za vsako navidezno pozicijo je shranjena številka dejanskega posla, odprtega na trgovalnem strežniku.

```
PosLink
    PosLink Version=0 Count=6
    102~1 #246096094 S*1 @o2011.11.07_22:00~0
    102~2 #246096076 S*2 @o2011.11.07_22:14~0
    102~3 #246098985 S*3 @o2011.11.07_22:36~0
    102~4 #246099070 S*4 @o2011.11.07_22:37~0
    102~5 #246099155 S*5 @o2011.11.07_22:38~0
    102~6 #246099476 S*6 @o2011.11.07_22:39~0
```

3.6.2 Varovanje pred napakami v algoritmu

V samem algoritmu za trgovanje lahko pride do napak, na primer odpiranja pozicij s prevelikimi količinami ali pa odpiranje in zapiranje pozicij z veliko večjo frekvenco, kot je pričakovano - zaradi napake v programu ali pa zaradi nepričakovanih pogojev na trgu. Knjižnica `PosLink` ima tri spremenljivke, s katerimi lahko omejimo izgubo, ki bi se zgodila zaradi takih napak:

- `FreeMargin` - koliko mora biti denarna rezerva na računu (angl. *free margin*) na trgovalnem računu, da se bo posel izvedel,
- `LotsPerHour` - kolikšna je lahko največja skupna vrednost poslov v zadnji uri,
- `LotsPerDay` - kolikšna je lahko največja skupna vrednost poslov v zadnjih 24 urah.

3.6.3 Redukcija izpostavljenosti

Za vsako strategijo, ki jo želimo sinhronizirati, lahko nastavimo parameter `Accelerator`, s katerim lahko zmanjšamo izpostavljenost poslov. Ima dva načina delovanja: izpuščanje poslov, ter sinhronizacija izpostavljenosti. Če je vrednost tega parametra 0, se ta nastavitev ne upošteva - sinhronizirajo se vse pozicije.

Pozitivna vrednost pomeni, da sistem izpušča posle. Dejansko jih izvede toliko odstotkov, kot je vrednost Accelerator med 0 in 1. Na primer vrednost 0,75 pomeni, da izvede 75% poslov.

Da se bo pri vrednosti *Accelerator* nek posel s številko *TradeID* izvedel, mora veljati:

```
MathFloor( ( TradeID - 1 ) * Accelerator ) == MathFloor( TradeID * Accelerator )
```

Primer: Pri vrednosti Accelerator = 0,4 bodo izvedeni samo posli z zaporednimi številkami: 1, 2, 4, 6, 7, 9, 11, 12, 14, itd.

Kadar pa nastavimo Accelerator na negativno vrednost, se posli odpirajo na tak način, da se prilagodi izpostavljenost na trgu. To je sicer manj pregledno, saj ni direktno razvidno kateri posli so odprti, je pa bolj primerna rešitev kadar je vrednost Accelerator majhna, na primer, ko želimo nastaviti izpostavljenost le na nekaj odstotkov.

Vsaka strategija poleg seznama navideznih pozicij vsebuje tudi štiri celoštevilske spremenljivke poimenovane Snake[5].

Te štiri vrednosti so poimenovane La, Lb, Sa, Sb. Dve od teh vrednosti, La in Lb odražata stanje dolgih poslov, Sa in Sb pa kratkih poslov oziroma navideznih pozicij. Kadar strategija odpre novo navidezno pozicijo, se poveča vrednost b za toliko, kot je velikost te pozicije, s čimer se daljica podaljša in posledično naj bi bila izpostavljenost večja. Kadar strategija zapre obstoječo navidezno pozicijo, se poveča vrednost a, s čimer se daljica skrajša, posledično pa naj bi bila izpostavljenost na trgu manjša.

Dejansko izpostavljenost, izraženo kot vrednost spremenljivke `exposure`, izračunamo s postopkom:

```
numLong =  MathFloor( Lb * Accelerator ) - MathFloor( La * Accelerator )
numShort = MathFloor( Sb * Accelerator )  - MathFloor( Sa * Accelerator  )
exposure = numLong - numShort;
```

Izpostavljenost je lahko pozitivna ali negativna vrednost. Pozitivna pomeni, da moramo imeti odprto dolgo pozicijo take velikosti, negativna pa da moramo imeti odprto kratko pozicijo.

[5] Video igra Snake, zaradi podobnosti koncepta, kjer imamo premikajočo daljico.

3.7 Knjižnica Sync

Vsebuje funkcije za nalaganje in shranjevanje stanja strategije (v datoteko ali HTTP strežnik) ter funkcije za sinhronizacijo simulacije.

Vhodni parametri so podani kot niz znakov (podatkovni tip `string`) in se preberejo s pomočjo knjižnice Assoc opisane v poglavju 3.8. Definiramo lahko naslednje parametre:

- `host` (niz znakov) - naslov HTTP strežnika, kjer se nahaja stanje strategije. Niz "`http://`" se samodejno doda, in ga ni potrebno vpisovati. Kadar vpišemo vrednost `file`, se bo vrednost shranjevala oziroma prebrala iz lokalne datoteke - to velja tudi, če te vrednosti ne definiramo ali pa podamo prazen niz.
- `cluster` (niz znakov) - kadar se shranjuje na HTTP strežnik, lahko opcijsko definiramo ime gruče, h kateri ta strategija spada, za lažjo organizacijo in večjo preglednost.
- `read` (logična vrednost) - definiramo, ali naj ob zagonu strategije preberemo obstoječe stanje.
- `write` (logična vrednost) - definiramo, ali naj zapisujemo spremembe stanja.
- `global` (logična vrednost) - definiramo, ali naj se seznam navideznih pozicij sinhronizira preko globalnih spremenljivk.

3.8 Knjižnica Assoc

Za enostavnejše podajanje parametrov različnim podsistemom ter interno dostopanje do spremenljivk smo naredili knjižnjico za delo z asociacijskimi tabelami. Asociacijska tabela je definirana kot 2-dimenzionalna tabela. Dolžina prve dimenzije je enako številu elementov v tabeli, dolžina druge dimenzije je 2 - ključ in vrednost vsakega elementa. Ker se vrednost shranjuje kot niz znakov, lahko predstavlja kateregakoli od osnovnih podatkovnih tipov.

Definirali smo naslednje funkcije:

- `AssocClear(string &a[])` - izbriše vse elemente v asociativni tabeli.
- `void AssocSet(string &a[][], string key, string val)` - postavi element, katerega ključ je enak parametru *key*, na vrednost *val*. Če element s takim ključem ne obstaja, ga ustvari.

- `int AssocIndex(string &a[][], string key)` - poišče element, katerega vrednost ključa je enak parametru *key* in vrne njegovo zaporedno vrednost v tabeli. Če tak element ne obstaja, vrne vrednost -1.

- `bool AssocContains(string &a[][], string key)` - pregleda tabelo, če v njej obstaja element katerega vrednost ključa je enak parametru *key* ter vrne logično vrednost *true*, oziroma $false$, če tak element ne obstaja.

- `string AssocGet(string &a[][], string key)` - vrne vrednost elementa, katerega ključ je enak *key*.

- `double AssocGetInt(string &a[][], string key)` - podobno kot AssocGet, pridobi in vrne vrednost pretvorjeno v tip `int` .

- `double AssocGetDouble(string &a[][], string key)` - podobno kot AssocGet, pridobi in vrne vrednost pretvorjeno v tip `double` .

- `string AssocToString(string &a[][])` - vrne tekstualno predstavitev asociativne tabele, katero je mogoče prebrati s funkcijo `AssocFromString`.

- `void AssocFromString(string &a[][], string str)` - iz parametra `str` prebere vrednosti in jih shrani v tabelo, podano kot spremenljivka `a` .

Primer tabele:

Ključ	Vrednost
address	“http://www.example.com”
withSpaces	“Vrednost s presledki”
numA	“123”
isActive	“”
frac	“0.048”

Tabela 3.1 Primer dvo-dimenzionalne tabele, ki jo lahko obravnavamo kot asociacijsko tabelo.

Tekstualna predstavitev z izvedbo funkcije `AssocToString` bi za tabelo 3.1 bila sledeča: “address=http://www.example.com withSpaces=”Vrednost s presledki”numA=123 isActive frac=0.048”

4 Vzorčni algoritem "Simplex"

Za prikaz delovanja dodanih funkcionalnosti smo izdelali preprost algoritem, ki na podlagi indeksa relativne moči (angl. *relative strength index*) vstopa v posle. Pogoj za vstop v kratko pozicijo je, da mora biti vrednost RSI pod neko vrednostjo, ki jo definiramo kot parameter `RsiTreshold`. Za dolge pozicije velja podobno, le da mora biti vrednost indikatorja RSI nad `100 - RsiTreshold`. Zaprtje pozicije je pogojeno z njeno starostjo, to pa lahko nastavimo s parametrom `TradeMaxAge` – ko je neka pozicija odprta toliko svečk, kot je nastavljena ta vrednost, se pozicija zapre. To je primer metode statističnega trgovanja, s katerim poizkušamo izkoristiti kratkotrajne spremembe cene, namesto da bi trgovali s trendom.

Ker želimo to strategijo izvajati na več valutnih parih skupaj, je potrebno zmanjšati tveganje ob nestabilnih obdobjih na trgu, ko se menjalni tečaji dalj časa premikajo v eno smer. V takih situacijah bi ta strategija odprla veliko število pozicij, kar pa želimo omejiti. Za namen komunikacije med primerki strategij smo definirali skupno spremenljivko z imenom `NumTrades`, ki odraža skupno število odprtih pozicij na vseh valutah skupaj. Pogoj za odprtje pozicije pa nekoliko prilagodimo: za odpiranje kratke pozicije mora

biti indikator RSI pod izračunano vrednostjo `=(RsiTreshold-NumTrades)`, pri kratki dolgi pa mora biti indikator RSI na vrednostjo `=(100-(RsiTreshold-NumTrades))`. To je primer strategije, ki jo z običajno funkcionalnostjo MT4 ne moremo simulirati na zgodovinskih podatkih.

4.1 Izvorna koda

Za izdelavo strategije, ki uporablja funkcionalnosti naše nadgradnje, najprej ustvarimo datoteko `SimplexEA.mq4`, ki je datoteka vrste ExpertAdvisor.

```
//Definiramo parametre strategije, ki pogojujejo njeno delovanje
extern double    RsiTreshold = 0;
extern int       TradeMaxAge = 0;

//Vključimo podporne funkcije v načinu "ExpertAdvisor"
#include "indicators/kepa/base.ea.mqh"

//Vključimo datoteko s kodo trgovalne logike.
#include "indicators/logic/Simplex.logic.mqh"
```

Če želimo izvajati več primerkov te strategije na istem grafu s ceno, lahko zanjo definiramo tudi datoteko vrste Indicator. Ime te datoteke je `Simplex.mqh`. Razlika z zgornjo kodo je le v tem, da namesto vključevanja datoteke `base.ea.mqh` vključimo datoteko `base.indicator.mqh`. Pri tem pazimo na pot do vključenih datotek, saj je ta datoteka že v mapi `indicators`. Pred parametri ni ukaza `extern` saj v načinu indikatorja vrednosti parametrov dobi preko niza znakov, ki je definiran v podpornih funkcijah.

```
//Definiramo parametre strategije, ki pogojujejo njeno delovanje
double     RsiTreshold = 0;
int        TradeMaxAge = 0;

//Vključimo podporne funkcije v načinu "Indicator"
#include "kepa/base.indicator.mqh"

//Vključimo datoteko s kodo trgovalne logike.
#include "logic/Simplex.logic.mqh"
```

Sledi še datoteka s trgovalno logiko, ki je nekoliko obsežnejša, ampak zaradi vseh podpornih funkcij vsebuje le odločitve glede odpiranja in zapiranja poslov. Začetek te

datoteke vsebuje osnovne definicije. Ime datoteke je `Simplex.logic.mqh`.

```
//Ime strategije - za beleženje in prikaz v uporabniškem vmesniku.
string StrategyName() { return ( "Simplex" ); }

//Funkcija, ki shrani vrednosti parametrov, ki so prebrani iz niza
//znakov v načinu delovanja kot "Indicator".
bool ApplyParam( string name, double value )
{
   if ( name == "RsiTreshold" )
   { RsiTreshold = value; return ( true ); }
   if ( name == "TradeMaxAge" )
   { TradeMaxAge = value; return ( true ); }
   return ( false );
}

//Izpis morebitnih trenutnih spremenljivk, ki so lahko v pomoč pri
//preverjanju pravilnega delovanja
string ParamsToString() { return ( "[...]" ); }
string EnvironmentToString(){ return( "" ); }

//V metod Initialize() definiramo minimalno število svečk, ki morajo
//biti na grafu, da se lahko strategija izvaja
void Initialize(){ RequiredCandles( 100 ); }

//Sledeče funkcije v tem primeru niso uporabljene, še vseeno pa morajo
//biti definirane, da se koda prevede.
void Configure(){}
void Dispose(){}
bool OnEvent( int etype ){}
void OnTick( int c ){}
```

Sledi programska koda, ki izvaja trgovalno strategijo. Sledeča funkcija `OnCandle` se izvede, kadar se na grafu pojavi nov časovni interval oziroma t.i. *svečka*.

```
void OnCandle( int c )
{
   //Pogoji za odpiranje pozicij
   double rsi = iRSI ( NULL, NULL, 14, PRICE_OPEN, c );
   double tholdLong = GetTreshold();
   double tholdShrt = 100 - tholdLong;
   int fi = GetFamilyIndex( LONG );

   if ( rsi < tholdLong ) {
      //Ustvarimo novo kratko pozicijo
      PosAdd( GetFamilyIndex( SHRT ), Time[c], Open[c] );
      SharedVariableInc( "NumTrades", 1 );
   }
   if ( rsi > tholdShrt ) {
      //Ustvarimo novo dolgo pozicijo
      PosAdd( GetFamilyIndex( LONG ), Time[c], Open[c] );
      SharedVariableInc( "NumTrades", 1 );
   }

   int ageSeconds, ageCandles;
   //V zanki se sprehodimo čez vse pozicije.
   for( int pi = ArrayRange(P,0)-1; pi >= 1 ; pi-- ) {
      //Izračunamo starost pozicije.
      ageSeconds = Time[c] - P[pi, VIPO_OTIME];
      ageCandles = ageSeconds / ( Period() * 60 );

      //Če je starost dosegla vrednost parametra TradeMaxAge, jo zapremo.
      if ( ageCandles >= TradeMaxAge ) {
         SharedVariableInc( "NumTrades", -1 );
         PosRemove( pi );
         pi--;
      }
   }
}
```

Vsaka navidezna pozicija mora spadati k neki skupini pozicij. Te skupine smo poimenovali *družine*. Zaradi boljše preglednosti trgovalne logike smo delo s skupinami napisali v ločeni metodi. Prav tako smo ločili kodo, ki iz modula na HTTP strežniku

`SharedVariables` pridobi vrednost skupne spremenljivke z imenom `NumTrades`.

```
//Metoda ustvari novo skupino pozicij, dolgih ali kratkih, glede na
//vrednost parametra op. Za dolge pozicije je OP=0, za kratke OP=1
int GetFamilyIndex( int op )
{
   int fi = -1;
   for( int i = 1; i < ArrayRange(F,0); i++ )
      if ( F[i,VIFA_OP] == op ) { fi = i; }
   if ( fi == -1 )
      fi = PosFamilyAdd( op );
   return ( fi );
}

//Metoda iz strežnika pridobi vrednost skupne spremenljivke "NumTrades"
// in jo odšteje odparametra RsiTreshod, da dobimo trenuten
// prag vrednosti indikatorja RSI za odpiranje pozicij
double GetTreshold()
{
   double numTrades;
   SharedVariableGet( "NumTrades", numTrades );
   return ( RsiTreshold - numTrades );
}
```

4.2 Simulacija strategije

Za najboljši prikaz delovanja te strategije smo si izbrali simulacijo na podatkih med datuma 1. in 15. februar 2014 na valutnih križih EURUSD in USDCHF. Ker je vrednost švicarskega franka trenutno izenačena z evrom [17], je gibanje tečaja teh dveh valutnih križev zelo podobno, pravzaprav obratno oziroma popolnoma negativno korelirano.

Seznam parametrov testa:

- Vrednost parametra `RsiTreshold`: 20,
- Vrednost parametra `TradeMaxAge`: 15,
- Instrumenti: EURUSD, USDCHF,
- Vir podatkov: Alpari UK Ltd.,
- Cenovni razmik (angl. *spread*): 2 točki,

- Začetek obdobja: 1. februar 2014, 0:00 GMT,

- Začetek obdobja: 15. februar 2014, 0:00 GMT.

Pričakujemo, da bo podobno gibanje obeh valutnih križev ustvarilo situacije, kjer bi oba primerka te strategije vstopala v posel. Toda ker vodimo skupno spremenljivko NumTrades, ki spreminja pogoj vstopa, je vstop manj verjeten, če imamo več odprtih poslov. S tem poizkušamo zmanjšati tveganje.

4.2.1 Razlaga merilnikov uspešnosti testiranja

Pri izvedbi simulacije nas za določeno strategijo pri nekih vrednostih parametrov zanimata predvsem končni dobiček oziroma izguba (angl. *total profit*) ter največji padec denarnega stanja tekom izvajanja (angl. *maximum drawdown*).

Pogosto pri testiranju različnih parametrov ugotovimo, da z večanjem končnega dobička raste tudi največji padec, torej bolj kot so neke vrednosti parametrov dobičkonosne, bolj se povečuje tveganje. Zato se uporablja tudi razmerje med dobičkom in padcem (angl. angl. *profit-to-drawdown ratio*). Na primer, ob dobičku 1125$ in največjim padcem 500$, je razmerje dobiček/padec 2,25.

Število poslov je lahko pomemben dejavnik, ki pove zanesljivost rezultatov glede na morebitno spreminjanje trgovalnih stroškov, torej provizij oziroma cenovnega razmaka. Če tekom simulacije algoritem odpre majhno število poslov, ki ustvarijo velik dobiček, potem povečanje trgovalnih stroškov ne bi bistveno vplivalo na rezultat. Kadar pa algoritem naredi veliko število poslov, ter vsak posel naredi zelo majhen dobiček, potem lahko majhna sprememba trgovalnih stroškov zelo vpliva na končen rezultat.

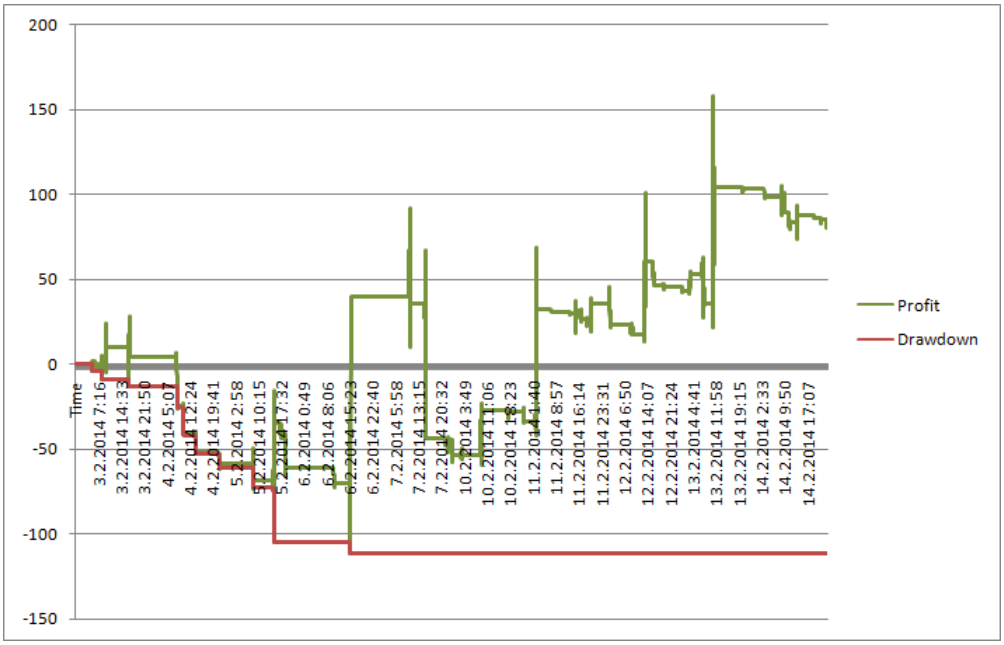
4.2.2 Izvedba v neodvisnem načinu

Za izvedbo te simulacije smo izvorno kodo trgovalne logike spremenili tako, da skupna spremenljivka `NumTrades` ni več ista za oba primerka, ampak ima vsak instrument svojo – `NumTradesEURUSD` ter `NumTradesUSDCHF`.

Tabela 4.1 prikazuje številčne rezultate testa, na sliki 4.1 pa je razviden graf gibanja denarnega stanja med izvajanjem algoritma. Opaziti je mogoče, da skupni padec ni enak seštevku padcev testa na obeh instrumentih, kar pa ni napaka, saj se ta dva padca nista zgodila ob enakem času.

Instrument	Dobiček	Padec	Profit/Padec	Število poslov
EURUSD	61,10$	97,40$	0,63	72
USDCHF	23,02$	81,52$	0,28	84
Skupaj	84,12	151,48	0,56	156

Tabela 4.1 Rezultati testiranja v neodvisnem načinu.



Slika 4.1 Gibanje denarnega stanja med izvajanjem strategije v neodvisnem načinu.

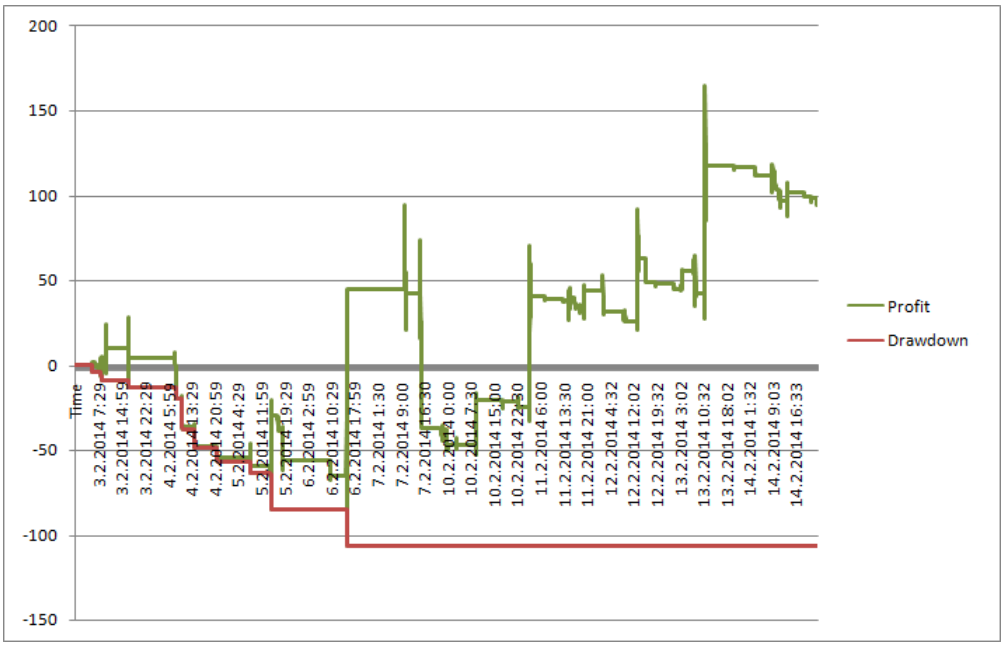
4.2.3 Izvedba v sinhroniziranem načinu

Simulacija se je izvedla v 14 minutah in 9 sekundah, kar je v primerjavi s klasično simulacijo, ki se izvede v manj kot sekundi, veliko počasneje.

Če primerjamo tabeli 4.1 in 4.2, je razvidno da je algoritem izvedel manj poslov v sinhronem načinu. Ideja zmanjševanja tveganja v tem primeru deluje, saj je bil dosežen večji dobiček ob manjšem padcu. Vrednost profit/padec je večja. Na sliki 4.2 si lahko ogledamo gibanje denarnega stanja med izvajanjem strategije, ter če jo primerjamo s sliko 4.1, lahko prav tako opazimo razlike.

Instrument	Dobiček	Padec	Profit/Padec	Število poslov
EURUSD	70,0$	97,40$	0,63	68
USDCHF	28,17$	70,94$	0,40	76
Skupaj	98,17	147,82	0,66	144

Tabela 4.2 Rezultati testiranja v sinhronem načinu.



Slika 4.2 Gibanje denarnega stanja med izvajanjem strategije v sinhronem načinu.

5 Zaključek

V diplomski nalogi smo preučili in opisali algoritemsko nadzorovano trgovanje. Opisali smo nekatere popularne metode tovrstnega trgovanja, torej načine, ki se med seboj razlikujejo in jih lahko ločimo v različne vrste. Pričakujemo lahko, da se bo delež takega trgovanja še povečeval ali pa vsaj obstal, ta trend pa lahko zaustavijo regulatorne spremembe. Ob povečevanju računskih zmogljivosti računalnikov ter dejstvu, da iz vidika tehnične analize že sedaj računalnik trguje boljše kot človek, je težko verjeti, da bo algoritemsko trgovanje kdaj postalo stvar preteklosti.

Poleg prednosti smo opisali tudi nekaj slabosti, kot je stalna prisotnost možnosti tehničnih napak, zaradi katerih trgovanja ne moremo popolnoma prepustiti računalniku. Poleg tega algoritmi še niso sposobni sami najti novih virov informacij in jih pravilno vključiti v odločanje.

Našteli smo nekatere programske rešitve, ki so dostopne tudi malim investitorjem. S temi lahko kdorkoli z nekaj računalniškega znanja (predvsem programiranja) in prostega časa izdela algoritem za izvajanje trgovalne strategije po lastni ideji. Tak algoritem je mogoče simulirati na zgodovinskih podatkih, nato pa že manjšim vložkom, naprimer

nekaj sto evri, tudi izvajati na pravem trgovalnem računu.

Ob tem ne smemo pozabiti, da moramo pri trgovanju s pravim denarjem upoštevati tudi vse nevarnosti trgovanja z vrednostnimi papirji, še posebej trgovanja z vzvodom. Izgubimo lahko ves vložen kapital, kadar uporabljamo vzvod pa celo več. Kljub navdušenju, znanju programiranja in mnogim idejam, se mora bodoči avtor algoritma trgovalne strategije zavedati, da je možnost neznanskih dobičkov praktično neobstoječa. Ne moremo pričakovati, da bomo z vložkom nekaj sto evrov obogateli v kratkem času. Lahko pa uspešno izdelamo algoritem, ki dosledno prinaša dobičke – dodatek mesečnemu dohodku ali pa alternativa klasični investiciji v delnice. Uspešen algoritem je mogoče tudi prodati ali dati v uporabo večjim investitorjem.

Nato smo se osredotočili na delovanje programskega paketa MetaTrader4 ter pri njem identificirali nekatere pomanjkljivosti. Z ustrezno programsko kodo smo poenostavili izdelavo trgovalne logike, torej programske kode, ki določa vstopanje v posle ter njihovo zapiranje. S pomočjo tega sistema programerju trgovalne logike ni več potrebno pisati dodatne kode, naprimer za prestrezanje napak in shranjevanje spremenljivk, temveč se lahko osredotoči samo na pravila strategije – programske kode je tako manj in je bolj pregledna.

S to nadgradnjo smo omogočili tudi to, da lahko izvajamo več simulacij trgovalnih strategij hkrati, naša koda pa poskrbi za sinhronizacijo simulacije. To je ključnega pomena takrat, kadar se strategije medsebojno povezujejo. Ločitev logičnega dela od sistemskega pa je hkrati tudi temelj za izvajanje nekega algoritma na več računalnikih hkrati, s čimer lahko omogočimo redundančnost. Delovanje naše nadgradnje smo prikazali v poglavju 4, kjer opišemo izdelavo trgovalne strategije, pri kateri je pogoj vstopanja v posel direktno povezan s trenutno izpostavljenostjo ostalih strategij – več kot je odprtih poslov, manj verjetneje je, da bo nek primerek tega algoritma vstopil v nov posel.

Programska koda v jeziku MQL skupaj z vzorčnim algoritmom zavzema 128 kilobajtov. V 27 datotekah z izvorno kodo je skupaj 4690 vrstic programske kode in 330 vrstic komentarjev. Izvorna koda HTTP strežnika v jeziku C# obsega 2697 vrstic programske kode ter 101 vrstic komentarjev. Za štetje vrstic kode smo uporabili program CLOC [18].

Ob testiranju smo ugotovili, da je sinhrono simuliranje veliko počasnejše kot izvajanje normalnega testa. Za pohitritev bi lahko namesto HTTP oziroma TCP/IP protokola uporabili t.i. imenovane cevi (angl. *named pipes*), toda potem bi izgubili možnost, da za izvajanje simulacije uporabimo več računalnikov hkrati.

Ugotovili smo, da je programski jezik MQL4 precej omejen. Ne podpira mnogih funkcionalnosti, ki so značilne za modernejše programske jezike. Nabor vgrajenih funkcij je majhen, torej smo mnoge morali narediti sami, kot na primer funkcije za delo s tabelami. Lahko, da bi bila boljša alternativa izdelava samostojnega programa v modernejšem in bolj razvitem programskem jeziku, v MT4 pa bi izdelali ExpertAdvisor, ki bi deloval kot API vmesnik za pridobivanje novih podatkov ter posredovanje poslov. To rešitev bi lahko izvedli tudi s pomočjo klicev funkcij v DLL datotekah, kar MT4 podpira.

Vsekakor smo spoznali obsežno področje trgovanja na borzah, ki ne bi bilo mogoče brez računalnikov in računalniških znanj. Poleg tega pa smo temu področju tudi dodali uporaben kos programske opreme.

Literatura

[1] Wikimedia Foundation. Fundamental analysis.
http://en.wikipedia.org/wiki/Fundamental_analysis, February 2014.

[2] Wikimedia Foundation. Technical analysis. http://en.wikipedia.org/wiki/Technical_analysis, February 2014.

[3] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart, Gerald Tesauro. Agent-human interactions in the continous double auction.
http://spider.sci.brooklyn.cuny.edu/~parsons/courses/840-spring-2005/notes/das.pdf, 2001.

[4] Tyler Durden. Welcome to sub-nanosecond markets.
http://www.zerohedge.com/news/welcome-sub-nanosecond-markets, May 2012.

[5] Whitney Kisling. Knight capital reports net loss after software error.
http://www.bloomberg.com/news/2012-10-17/knight-capital-reports-net-loss-as-software-error-takes-toll-1-.html, October 2012.

[6] Graham Bowley. Lone $4.1 billion sale led to flash crash in may.
http://www.nytimes.com/2010/10/02/business/02flash.html, 2010.

[7] Dukascopy. Historical data feed.
http://www.dukascopy.com/swiss/english/marketwatch/historical, October 2012.

[8] Wikimedia Foundation. High-frequency trading.
http://en.wikipedia.org/wiki/High-frequency_trading, October 2012.

[9] Irene Aldridge. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems.* Wiley Trading, 2013.

[10] Dan Collins. High frequency trading: What's the big deal. http://dancollinsreport.com/high-frequency-trading-whats-the-big-deal/, 2013.

[11] Hasbrouck-Saar. Low-latency trading. http://ssrn.com/abstract=1695460, October 2010.

[12] Rich Miller. Hibernia plans high-speed trans-atlantic cable . https://www.datacenterknowledge.com/archives/2010/09/30/hibernia-plans-high-speed-trans-atlantic-cable/, 2010.

[13] Securities and Exchange Commission. Regulation of exchanges . http://www.sec.gov/rules/final/34-40760.txt, April 2000.

[14] MetaQuotes Software. MQL4 programming language documentation. http://docs.mql4.com/, October 2014.

[15] M. Pilato B. Collins-Sussman, B. Fitzpatrick. *Version Control with Subversion.* 2004.

[16] G. Held. *TCP/IP Professional Reference Guide.* 2001.

[17] Swiss National Bank. Swiss national bank sets minimum exchange rate at chf 1.20 per euro. http://www.snb.ch/en/mmr/reference/pre_20110906/source/pre_20110906.en.pdf, 2011.

[18] Al Danial. Cloc - count lines of code. http://cloc.sourceforge.net/, 2014.